INGENIERÍA DEL SOFTWARE

16/1/2020

Desarrollo de un motor de búsqueda y almacenamiento de contenidos con Apache Solr

AUTOR: MARCOS RIVEROLA ERRANDO

DIRECTOR: GUILLERMO ORTIZ FIGUEROA

PONENTE: CARLES FARRÉ TOST

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

Facultat d'Informàtica de Barcelona

FIB

Resumen

Hoy en día, mantener la información estructurada y accesible es muy importante para una empresa. La empresa donde he realizado este Trabajo de Final de Grado, tiene como cliente una de las empresas de negocios más importantes de Barcelona.

Este proyecto tiene como objetivo poner solución a diversos problemas de cara a la centralización, estructuración y accesibilidad de la información de dicha universidad.

Analizaremos todas las posibles soluciones y utilizando tecnologías punteras llevaremos a cabo una solución escalable y re aprovechable en un futuro.

Resum

Avui en dia, mantenir la informació estructurada i accessible és molt important per una empresa. La consultoria on he realitzat aquest Treball de Final de Grau, té com a client una de les empreses de negocis més importants de Barcelona.

Aquest projecte té com a objectiu posar solució a diversos problemes de cara a la centralització, estructuració i accessibilitat de la informació d'aquesta universitat.

Analitzarem totes les possibles solucions i utilitzant tecnologies punteres portarem a terme una solució escalable i re aprofitable en un futur.

Abstract

Nowadays, keeping the information well-structured and accessible is something really important for a company. The consultancy where I have done this Final Degree Project has as a client one of the most successful business schools of Barcelona.

This project has as goal getting to a solution to some problems towards the centralization, structure and accessibility of the information of this university.

We will analyze all the possible solutions and using cutting-edge technologies we will find a scalable and reusable solution.

Agradecimientos

Agradezco a TBSCG por haberme brindado la oportunidad de llevar a cabo este proyecto tan interesante.

A mi tutor Carles que me ha ayudado durante todo el proceso con sus correcciones.

A mis compañeros con los que nos hemos animado unos a otros en momentos complicados para tirar adelante.

Y sobre todo a mi familia que cada vez que dudaba me hacían ver la realidad y me han apoyado durante toda la carrera.

Contenido

1 Contexto

1.1 Introducción

Este proyecto consiste en un Trabajo de final de grado de la Facultad de Informática de Barcelona. Mas concretamente de la especialidad de Ingeniería del Software.

El objetivo principal del proyecto es la creación de un motor de búsqueda de contenidos de un portal web para una universidad. Se busca proporcionar una solución eficiente, robusta y funcional para resolver un problema el cual es la búsqueda de contenidos específicos cuando se dispone de tanta información.

1.2 Descripción del proyecto

Desde The Banyan Solutions Consulting Group (TBSCG) empresa en la cual he emprendido este trabajo de final de grado, ofrecemos nuestros servicios de consultoría para una universidad de Barcelona. Dicha universidad había hecho recientemente un cambio de portal web, que estaba construido por un WordPress. Tenían un claro problema: necesitaban centralizar todos sus contenidos dispersados en diferentes fuentes en una sola base de datos y a la vez obteniendo un potente motor de búsqueda.

Por estos motivos, se decidió utilizar Apache Solr como solución a estos problemas. Solr es un software libre destinado a las búsquedas distribuido por la empresa Apache. Este está basado en el proyecto Lucene también de Apache el cual es un software de recuperación de información basado en Java.

Solr nos permite volcar contenidos en él, estructurarlos, incluyendo funciones como la búsqueda por facetas o caché, lo que lo convierten en un software realmente potente. Solr tiene una interfaz para su administración a la cual se accede a través de una aplicación web, pero sin embargo tiene una gran integración con JAVA (ente otros, pero es en este lenguaje en el que está mayoritariamente desarrollado) para poder volcar contenidos de forma masiva y programada.

Después de presentar la tecnología principal, muy por encima, cabe resaltar que es un proyecto que aplica los conocimientos de la especialidad de Ingeniería del software, desde la toma de requisitos y el desarrollo hasta la integración y despliegue del software. Se utilizan metodologías y procesos estudiados en esta y pretende generar unos intereses a la empresa, así como resolver el problema por lo que la organización del proyecto es clave.

1.3 Stakeholders

Definimos Stakeholders como todas esas partes que están implicadas en el proyecto y que, o bien forman parte de él, o bien obtendrán un beneficio. [1]

Vamos a dividir los Stakeholders en dos grupos:

- Stakeholders primarios: Aquellos que están directamente relacionados con el proyecto y son imprescindibles para el funcionamiento de este o tienen una relación económica directa.
- Stakeholders secundarios: Aquellos que no participan directamente en las actividades de la empresa pero se ven afectados por esta.

El Stakeholders primario más claro será por supuesto el equipo de desarrollo de la empresa. Este está formado por una Jefa de Proyecto, encargada de la organización de tareas y comunicación con el cliente, un director técnico que supervisará la arquitectura del proyecto, así como el desarrollo de este y por último un desarrollador.

También identificamos en este grupo el cliente (la universidad), el cual invierte su dinero en el proyecto para obtener a cambio el producto final aun estando presente en todas las decisiones, pues ellos marcan los requisitos.

Una tercera empresa entra en este grupo, la empresa que desarrolla el portal web en WordPress, la comunicación con esta es fundamental ya que es la que desarrolla toda la parte Front-End a través de la cual se obtendrán los datos de nuestro Solr.

Pasando ya a los Stakeholders secundarios, encontramos principalmente a los usuarios del buscador: alumnos, profesores o potenciales alumnos de dicha universidad, entre muchos otros.

Así pues, será importante tener en cuenta estos agentes implicados en el momento de tomar según que decisiones, pues siempre hay que pensar para quien es el producto que desarrollamos.

1.4 Problemática

El problema principal de esta universidad es como se ha comentado anteriormente la dispersión de los datos, así como la falta de buscador en el portal.

Se disponen de muchísimos datos como, por ejemplo, artículos, profesores, eventos, becas para estudiantes, blogs, entre otros. Estos datos están distribuidos en diversos puntos. Algunos de estos están en bases de datos, otros incrustados en páginas HTML, o por ejemplo artículos en blogs personales de profesores.

Dicho esto, pasemos a identificar los problemas:

A. No existe una sola fuente de información la cual nos garantice la veracidad y actualidad de los datos que se muestran en el portal web.

B. Incapacidad de reaprovechamiento de la información de manera eficiente para otros portales dada la desestructuración de la información.

C. Diferentes métodos de obtención de dicha información desde el portal web dependiendo de su origen que implican codigo dispar y complicidad de reaprovechamiento de este.

D. Imposibilidad de listar y enumerar todos los contenidos del portal de forma eficiente ya que no provienen de un mismo sitio. Nunca se sabe la cantidad total de contenido.

E. Dificultad para aplicar filtros y organizar la información (busqueda por facetas).

F. Dificultad de relacion entre contenidos procedentes de diferentes origenes.

G. Inexistencia de código reusable en cuanto al buscador por lo que todo debe hacerse de zero, dado que antes no lo había.

H. Necesidad de un buscador de contenidos que devuelva los resultados pedidos aplicando filtros entre otros.

I. Falta de seguridad en el acceso a los contenidos y peticiones a estos.

J. Necesidad de busqueda fonetica dada la procedencia de algunos profesores que tienen nombres complicados de escribir y usualmente escritos mal.

K. Necesidad de customización al máximo posible de las respuestas del buscador en cuanto a orden, forzado de resultados y exclusión de estos.

2 Justificación

2.1 ¿Solución nueva o aprovechamiento?

Al empezar un proyecto siempre se nos plantea una pregunta inicial la cual hay que tener muy en cuenta, ¿Vale la pena empezar de nuevo y buscar una solución completamente nueva o mejor aprovechar el código y tecnologías disponibles y adaptarlos de forma que solucionemos los nuevos problemas?

En nuestro caso está pregunta no nos proporciona grandes dolores de cabeza pues es un proyecto completamente nuevo y no disponemos de ningún tipo de código, base de datos centralizada ni nada por el estilo. Por lo tanto, se llegó rápidamente a la conclusión de que hacía falta crear una solución nueva desde el principio.

La parte más complicada pues en nuestro proyecto era la decisión sobre que tecnologías usar, o que solución plantear. Dado que no había ninguna solución anterior no teníamos ningún tipo de guía sobre hacia qué dirección ir. Es por esto por lo que se plantearon diferentes opciones.

2.2 ¿Por qué Solr?

Ya hemos descrito previamente por encima que es Solr, sin embargo, no el por qué escogimos esta tecnología para solucionar los problemas planteados.

Bien, son varios los motivos por los que se decidió utilizar Solr como tecnología principal y el primero de estos es el hecho de que es Open Source. Siempre apostamos por tecnologías Open Source, y más aún si es una de las 2 más utilizadas en cuanto a motores de búsqueda. Esto nos permite ahorrar dinero obviando las soluciones comerciales, que más tarde podremos destinar a infraestructura, por ejemplo.

Otro gran motivo es la comunidad tan grande que tiene Solr detrás suyo. Al ser Open Source cualquiera puede aportar nuevas soluciones o funcionalidades. Realmente hay muchas personas y organizaciones detrás de esta tecnología que permiten la rápida solución de dudas lo cual se agradece siempre al empezar a usar una tecnología con la cual no estas familiarizado.

Solr es muy bueno en lo que hace: indexar contenido, organizarlo y recuperarlo más tarde. Algo sencillo pero efectivo, justamente lo que necesitamos.

Un motivo importante es su gran integración con el lenguaje de programación Java. No solo estamos muy familiarizados con este lenguaje, sino que además nos permite mucha flexibilidad en el momento de indexar contenidos. Esto se debe a que con un mismo proceso que veremos más adelante, somos capaces de recuperar todos los datos que están desperdigados por diferentes puntos de origen e indexarlos en nuestro Solr que actuará como base de datos y única fuente de veracidad.

Una funcionalidad muy importante de Solr es la posibilidad de subir un PDF y obtener estos datos a la vez que indexarlos. Esta funcionalidad es un punto muy fuerte para Solr dado que esta universidad posee muchos artículos los cuales deseaban indexar y organizar junto al resto de información.

Visto esto, podemos afirmar que Solr proporciona soluciones a la mayoría de nuestros problemas y es por eso por lo que lo escogimos como tecnología principal entre otros motivos.

2.3 Alternativas

Cuando hablamos de motores de búsqueda de software libre se nos vienen dos a la cabeza, Solr o Elasticsearch. Estos dos son los más utilizados por los desarrolladores, incluso más que aquellos de pago.

Haciendo un estudio intensivo de si utilizar uno u otro, llegamos a la conclusión de que Solr se ajustaba más a nuestras necesidades.

En este articulo [2] podemos ver las comparaciones. El hecho de que sea más complicado de instalar que elasticsearch no supone un problema para nosotros. Solr es más configurable en términos de ranking y nos permite mucha más customización que Elasticsearch.

Algo importante es que nos da la seguridad que nunca pasará a ser de pago dado que es trabaja bajo una licencia de Apache al igual que su padre Lucene. Elasticsearch sin embargo sí que es de la empresa Elastic, lo que podría derivar en una venta a otra empresa que decida empezar a cobrar por su uso.

Finalmente, la posibilidad de leer PDF y otros documentos hace que nos decantemos por Solr ante Elasticsearch.

3 Alcance

3.1 Objetivos principales

El objetivo principal del proyecto se podría dividir en 3 objetivos teniendo en cuenta lo mencionado anteriormente:

- Centralizar toda la información desperdigada por diferentes puntos de origen en una sola base de datos creando así una sola fuente fiable.
- Crear un motor de busqueda eficiente para recuperar toda esa información y customizarlo lo máximo posible.
- Dado que la información cambia, mantener siempre la información actualizada y mantenerla accesible y modificable para la universidad.

3.2 Objetivos específicos

Una vez identificados los objetivos principales podemos pasar a identificar los subobjetivos del proyecto:

- Analizar la cantidad de datos que hay que indexar en la base de datos así como los puntos de origen desde los cuales se obtendran.
- Analizar los recursos disponibles para poder determinar una infraestructura.
- Disenyar la solución de obtención de datos de diferentes puntos de origen en un solo proceso.
- Dicha solución deberá permitir la actualización de los datos de forma periodica y forzarla en un momento determinado si fuera necesario.
- Disenyar un sistema de seguridad para limitar el acceso y controlar las peticiones.
- Disenyar una serie de consultas a consumir por el Front-End para la recuperación de datos a través de la API de Solr.
- Implementar la aplicación manteniendo pruebas y comprobando el correcto funcionamiento de esta.

3.3 Requisitos no funcionales

Para un buen funcionamiento del sistema se deben definir también unos requisitos no funcionales que en nuestro caso son los siguientes:

- El sistema debe ser eficiente para que las respuestas sean lo suficientemente rapidas, por lo que las maquinas deben tener unas especificaciones mínimas.
- El sistema ha de estar disponible las venticuatro horas del dia los siete dias de la semana, exceptuando los despliegues que provocarán una caida de sistema momentanea y por lo tanto se harán en momentos de poco uso para minimizar daños.
- Se debe disponer de un entorno de pre producción para la aprovación del cliente de los cambios producidos antes de los despliegues a producción y que el contenido sea lo mas similar posible al entorno de producción.
- Se debe disponer de un sistema seguro dado que hay mucho contenido que debe ser protegido.

3.4 Obstaculos y riesgos

Uno de los riesgos principales es el desconocimiento de la tecnología que puede conllevar estancamiento en ciertos momentos, pero su gran comunidad es un punto a favor.

También podría suponer un riesgo el hecho de que las peticiones no son consumidas por nosotros sino por una empresa externa así que en cierta medida se depende de ella y de sus tecnologías a pesar de que nuestro deber es simplemente devolver datos a través de la API de forma segura.

Finalmente, podríamos considerar como un riesgo el hecho de que los nombres de algunos profesores son muy confusos y hay que tener en cuenta si o si la fonética. Así como tener en cuenta que la plataforma es multi idioma y habrá que trabajar teniendo en cuenta esa característica.

4 Metodología y rigor

4.1 Metodología waterfall

La metodología escogida para este proyecto es la conocida como waterfall [3] o cascada en español. Esta metodología consiste en una serie de fases que se han de ir completando para poder avanzar a la siguiente, emulando como su nombre indica una cascada de sucesos.

El porqué de la elección de esta metodología es por el hecho de ser un equipo tan pequeño con un solo desarrollador. Aun siguiendo esta metodología se adoptan costumbres o características de otras como las denominadas como Agile. Esto se debe a que hay una comunicación constante con el cliente, este siempre está al corriente de los cambios que se ejecutan y en las tomas de decisiones.

Otra característica de la metodología waterfall es que se acuerda un precio con el cliente, el cual no aumentará por el hecho de que el desarrollo será el acordado, ni más ni menos, hasta conseguir los objetivos establecidos. Si se decide introducir algún cambio a posteriori, alguna funcionalidad nueva, se tendrá que negociar y crear un nuevo presupuesto. Es una metodología muy robusta cuando se tienen objetivos muy claros y se sabe que estos no van a cambiar.

A pesar de que hoy en día las metodologías Agiles, en particular Scrum, están muy de moda, hemos decidido descartarla. Scrum consiste en un proceso incremental. Se construye, se recibe feedback del cliente y se itera sobre ese feedback. Aunque si mantendremos reuniones periódicas, tenemos muy claro el producto que hay que realizar. Los objetivos están muy definidos, por lo que cual un proceso secuencial, en este caso siguiendo la metodología Waterfall, se adapta perfectamente a nuestras condiciones. Una vez decididas las tecnologías a utilizar y sabiendo que cubren nuestras necesidades por completo, podemos confirmar que Waterfall es la opción correcta para trabajar en este proyecto.

Las fases de la metodología waterfall son las siguientes:

- Toma de requisitos: en la cual se recopilan los requisitos del software a desarrollar.
- Diseño del sistema: se discute la infraestructura, las tecnologias a utilizar, patrones de diseño y estrategias. En este punto aun no se ha empezado a desarrollar.
- Implementación: consiste en el desarrollo del producto, mucha gente lo liga con la siguiente fase la cual es la prueba del producto, aunque a nivel de desarrollador (hay muchas variantes de la metodologia waterfall que trabajan con equipos de prueba).
- Prueba: consiste en probar el producto una vez está desarrollado, sin miedo a obtener bugs o fallos. El objetivo es perfeccionar el producto y si hace falta volver dos fases atrás, rediseñar para luego volver a implementar.
- Entrega y despliegue: en esta fase se entrega el producto al cliente para que sea desplegado y llevado a producción donde los usuarios finales podran utilizarlo.
- Mantenimiento: la mayoria de proyectos cuentan con un mantenimiento final en que cuando el producto está en uso, el cliente levanta una incidencia y esta se soluciona (incluyendo rediseño e implementación si es necesario).

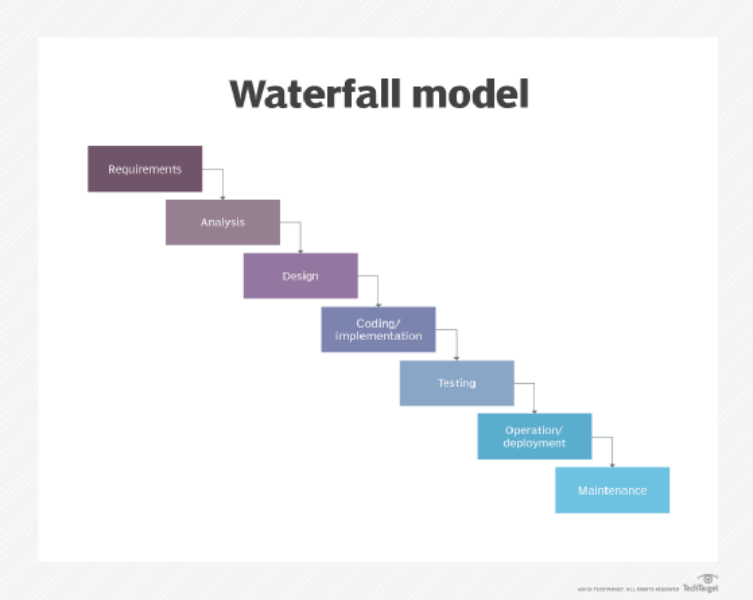

Figura.1 Metodología Waterfall

4.2 Herramientas

Se utilizarán diversas herramientas para la organización del proyecto. Estas son imprescindibles para el éxito de este.

La primera sería un repositorio GIT para el control de versiones que nos permitirá gestionar el código.

Como organizador de tareas se utiliza Jira el cual nos permite crear tableros para crear estimaciones de tareas, estado de estas y organizarlas para mantener un control en todo momento.

Una parte importante del proyecto es la documentación por lo que se utiliza Confluence. Confluence permite crear y organizar diversas páginas con sus apartados y varias funcionalidades (tablas, graficas, etc.) muy útiles para documentar los progresos del proyecto. La documentación es imprescindible en el caso de que algún desarrollador durante el proyecto cambie (aunque no sea el caso de este proyecto) para así saber exactamente que hay hecho y como. También se puede entregar al cliente como manual de uso y para que sepa exactamente que hay hecho y como.

Además de Confluence, como sabemos que hay una serie de llamadas a nuestra API que serán consumidas por el Front-end de una empresa externa, es importante mantener la documentación de estas llamadas actualizada. Es por eso por lo que generaremos un documento con la herramienta Postman, para que se puedan tener todas estas llamadas al alcance.

Para la comunicación con el cliente se utilizará Skype para las reuniones cuando sean necesarias y un canal de Microsoft Teams en el que consultar dudas con más frecuencia si se necesita.

5 Planificación de tareas

En este apartado veremos la planificación de las tareas a lo largo del proyecto. Veremos las diferentes fases de este, las tareas concretas de cada una con sus estimaciones, así como los recursos tanto humanos como materiales.

5.1 Tareas y fases del proyecto

En todo proyecto es necesario dividir este en fases para así poder organizar todas las tareas existentes. Independientemente de la metodología, esto es algo que se da siempre pues la organización es crucial y ayuda tanto a agilizar el proyecto como al entendimiento de este para todas las partes implicadas.

5.1.1 Tareas de gestión

Las tareas de gestión en este proyecto no se consideran una fase como tal, sino que irán siendo ejecutadas a lo largo de este. Formarán parte del proyecto en todo momento y tienen la misma importancia que cualquier otra. Así pues, definimos las siguientes tareas de gestión (TG):

-TG1: Reuniones semanales. Cada dos semanas nos reuniremos con el cliente para así poder enseñar los progresos, preguntar dudas existentes (de cara al futuro desarrollo, todas las dudas bloqueantes han de ser preguntadas en el momento) así como mantener la relación con este.

-TG2: Documentación. Es vital tanto en este como en cualquier proyecto. La documentación incluye la planificación, el alcance, el presupuesto, la memoria del proyecto y el manual de usuario. En ocasiones se hace como una tarea al final del proyecto, pero en este caso se hará durante este.

5.1.2 Fase de toma de requisitos

La fase de toma de requisitos (TR: tarea de requisitos) es la primera y tiene una importancia muy elevada. En esta fase se ven implicadas ambas partes y es vital que colaboren al unísono y fluidamente para llegar a un acuerdo.

El objetivo principal de esta fase es establecer básicamente que funcionalidades, así como el alcance del proyecto. Podríamos dividirlo en diferentes sub-fases las cuales se pueden llevar a cabo concurrentemente:

- TR1: La primera sería identificar los problemas a resolver. El cliente deberá mostrar a nuestra empresa cuales són los problemas a resolver y cualquier tipo de prioridad así como restricción.
- TR2: La segunda sería definir las funcionalidades que darán solución a dichos problemas. En este momento se podrían definir también las tecnologías a utilizar y proporcionar una estimación tanto temporal como económica total del proyecto (siempre que sea un proyecto de precio cerrado).

5.1.3 Fase de diseño

La fase de diseño se basa en la búsqueda de cómo se van a llevar a cabo las funcionalidades escogidas previamente en la toma de requisitos. Mayoritariamente se decide la arquitectura del proyecto, se deciden patrones de diseño a aplicar y se toman decisiones importantes, puesto que la arquitectura y diseño de un proyecto marcan su desarrollo. Sin una buena fase de diseño es imposible tener un buen desarrollo.

Por lo tanto, definimos la tarea de diseño (TD) TD1. En está queda involucrado el líder técnico, aunque con soporte del equipo de desarrollo. El líder técnico siempre será el que tomé la decisión final pero el equipo de desarrollo tiene que opinar por lo menos, pues será este quien tome las riendas en la siguiente fase y ha de sentirse cómodo con las decisiones tomadas.

Una vez decidida la arquitectura, se presentará una solución al cliente en la prueba de concepto. Una vez el cliente acepte, se comenzará el desarrollo.

5.1.4 Fase de implementación

La fase de desarrollo consiste en el desarrollo del código del proyecto, es por eso por lo que empezaremos a hablar de tareas de implementación (TI) mucho más concretas con estimaciones en horas.

TI1: Preparación de los entornos: Instalación de dependencias, preparar las instancias y descargar los programas necesarios. Asegurar servidores de producción y preproducción con certificados SSL [4] y autenticación http [5] básica. 32h

TI2: Creación en Solr de los diferentes índices donde se guardará la información los cuales serán: Antiguos alumnos, Blogs, Centros y Cátedras, Eventos, Ayudas Financieras, Clubs, Profesores, Programas, Publicaciones y Blogs. Hay que decidir la estructura de datos de cada uno de estos y sus configuraciones. 80h

TI3: Implementación del indexador de contenido para cada uno de los índices nombrados anteriormente. Consiste en obtener los datos de cada una de las fuentes de origen, tratar los datos y enviarlos a la instancia de Solr correspondiente (instancia local, preproducción y producción).160h

TI4: Habilitar búsquedas. Debemos habilitar diferentes búsquedas para cada uno de los índices. Cada uno de estos tiene distintos órdenes, filtros y por lo tanto cada una de estas ha de ser personalizada y eficiente. 120h

TI5: Funcionalidades genéricas de Solr. Desarrollo de distintas funcionalidades, independientes a los índices. Son diferentes características que ofrece Solr como por ejemplo la elevación de resultados, la corrección gramatical o detección de lenguaje automática. 50h

TI6: Despliegues a producción: Consiste en llevar el código desarrollado hasta el momento a un entorno de producción. Se hará varias veces durante el desarrollo y finalmente un despliegue final. 24h

5.1.5 Fase de pruebas

Una vez desarrollado el proyecto y entregado, se pasará a la fase de prueba en que el cliente usará la plataforma de forma normal antes de hacerse pública, buscando la detección de errores para poder corregirlos. Se harán 2 fases, la primera será después del despliegue a preproducción y la segunda después del despliegue a producción. La llamaremos tarea de pruebas (TP)

5.1.6 Fase final

Una vez el cliente ya ha probado todas las funcionalidades y se han ejecutado los errores necesarios, pasaremos a entregar el código final. La identificaremos como cierre de proyecto (CP).

Tarea	Descripción	Estimación(h)	Dependencias
TG1	Reuniones	24	
TG2	Documentación	120	
TR1	Definición de problemáticas	40	
TR2	Definición de funcionalidades	40	TR1
TD1	Diseño de arquitectura	40	TR2
TD2	Prueba de concepto	20	
TI1	Preparación de Entornos	32	TD1
TI2	Creación de índices	80	TD1
TI3	Implementación de Indexador	160	TI2
TI4	Implementación de búsquedas	120	TI3
TI5	Funcionalidades genéricas	50	TI2, TI3
TI6	Despliegues	24	TI1, TP
TP	Fase de pruebas	80	TI2
CP	Cierre de proyecto	16	TI6, TP

Figura. 2 Tabla de tareas con estimaciones.

5.2 Diagrama de Gantt

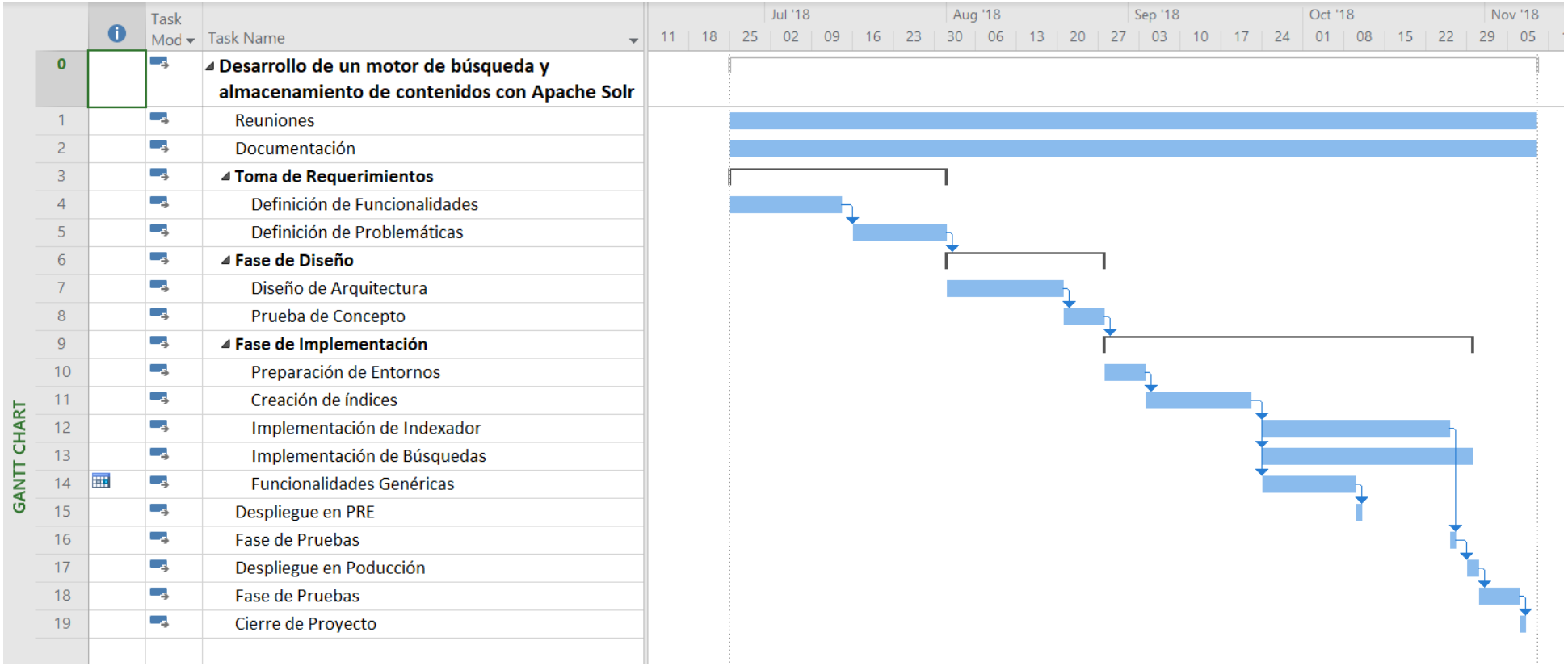

Figura 3. Diagrama de Gantt

5.3 Recursos

Diferenciamos entre dos tipos de recursos: los humanos y los materiales. Los humanos los podríamos clasificar en:

- Project Manager: define la dirección, gestiona plazos y recursos, gestiona la comunicación interna.

- Arquitecto y líder técnico: Asegura que la solución técnica se adapta a los requerimientos de negocio, permite la comunicación entre el cliente y el equipo de desarrollo, valida las pruebas y despliegues y da soporte al equipo de desarrollo.
- Equipo de desarrollo: implementa el código que va a dar soluciones a los problemas de la empresa cliente.

En cuanto a recursos materiales encontramos:

- Entorno de preproducción: 8 cores, 16 gb RAM, 50 gb de disco sas.
- Entorno de producción:
    - Maquina 1: 8 cores, 16 gb ram, 50 gb de disco sas.
    - Maquina 2: 8 cores, 32 gb ram, 100 gb de disco ssd.
- Java 1.8 junto con la librería SolrJ para el indexador.
- Solr 7.5.
- Nginx http server.
- Crontab scheduler.
- Postman.
- IntelliJ Idea.
- Git versión control

5.4 Gestión de riesgos

Como en todo proyecto existen ciertos riesgos que hay que estudiar y ofrecer alternativas para así anticiparse a los problemas. El estudio de estos es importante para saber que repercusión tienen en nuestro proyecto a nivel de tiempo y coste.

El primer obstáculo que encontramos es la participación de una tercera empresa que controla el front-end de nuestro proyecto. Por lo que tenemos una pequeña dependencia en cuanto al testeo de nuestras búsquedas. Es por

eso por lo que será necesario acordar previamente antes de hacer las búsquedas todas las características de estas: parámetros de entrada, respuestas esperadas y mensajes de error. De esta forma, nos aseguramos de que con herramientas como Postman podemos probar nuestras llamadas http y así saber que estamos cumpliendo nuestra parte. Nos supondrá tiempo de testeo de nuestras búsquedas, pero nada a nivel de coste pues Postman es una herramienta gratuita.

Como ya se comentó, un gran riesgo son los nombres de los profesores, los cuales crean mucha confusión en las búsquedas. Por eso se decide emplear tiempo en crear una búsqueda fonética. Es una funcionalidad que Solr incluye en su producto de forma gratuita (dado que es Open Source) y además utilizamos un repositorio de nombres y apellidos europeos en nuestras búsquedas para simplificar esta tarea.

Un gran riesgo podría ser el cambio de la estructura de datos en el origen. Esto puede suponer un elevado coste a nivel de tiempo dado que habría que cambiar tanto la estructura del índice, como la del indexador con el que se comunican ambos puntos. Habrá que hacer un código adaptable y muy poco acoplado para que pueda someterse a cambios si fuera necesario.

Solr es un sistema Open Source [6], por lo que puede dejar de avanzar en cualquier momento. Aunque es algo poco probable, siempre podría pasar. Lo crucial es saber que con la versión actual podemos solucionar todos nuestros problemas y pensar a futuro si fuera necesario. En el caso de encontrar un problema que Solr no nos pudiera solucionar habría que invertir cierto tiempo en investigación y utilizar alguna herramienta externa.

El último obstáculo es el uso del multi idioma. Se decide duplicar el contenido en cada uno de los indicies y añadir una propiedad lenguaje por la cual filtrar, la cual hace muy sencillas las búsquedas. Además, Solr nos ofrece una detección de idioma automática en el contenido por lo que no nos hemos de estar preocupando de este ya que el propio sistema nos lo clasificará.

6 Presupuesto

6.1 Identificación de los costes

6.1.1 Recursos humanos

En este proyecto toman partida varias personas las cuales ocupan un rol distinto cada una. Puede que una persona ocupe dos roles distintos (por ejemplo, el líder técnico puede ser el arquitecto). Teniendo esto en cuenta pasamos a calcular el presupuesto para nuestro proyecto.

Todos los precios por hora se han calculado dividiendo el salario anual entre las horas estipuladas en el convenio de oficinas y despachos que son 1772 horas. [7]

Para el cálculo de recursos humanos utilizaremos una tabla la cual contendrá: Tarea a realizar, horas de la tarea, personal que la realiza y total del coste de la tarea. Para eso nos hace falta aclarar los roles y el coste por hora de cada rol:

- Project manager: 25.40 euros/hora. [8]
- Desarrollador: 19.46 euros/hora. [9]
- Tester: 20.87 euros/hora. [10]
- Arquitecto: 22.89 euros/hora. [11]

Tarea	Horas de la tarea	Participantes	Total
Reuniones	24 horas	Project Manager (100%) Arquitecto (100%)	1158.96 euros
Documentación	120 horas	Project Manager (20%) Desarrollador (70%) Arquitecto (10%)	2518.92 euros

Definición de problemáticas	40 horas	Project Manager (100%)	1016 euros
Definición de funcionalidades	40 horas	Arquitecto (100%)	915.6 euros
Diseño de arquitectura	40 horas	Arquitecto (100%)	915.6 euros
Prueba de concepto	20 horas	Project Manager (90%) Arquitecto (10%)	502.98 euros
Preparación de entornos	32 horas	Arquitecto (100%)	732.48 euros
Creación de índices	80 horas	Arquitecto (10%) Desarrollador (90%)	1584.24 euros
Implementación de Indexador	120 horas	Arquitecto (10%) Desarrollador (90%)	2101.68 euros
Implementación de búsquedas	120 horas	Arquitecto (10%) Desarrollador (90%)	2379.36 euros
Funcionalidades genéricas	50 horas	Arquitecto (10%) Desarrollador (90%)	990.15 euros
Despliegues	24 horas	Desarrollador (100%)	467.04 euros
Fase de Pruebas	80 horas	Desarrollador (35%) Tester (65%)	1630.12 euros
Cierre de proyecto	16 horas	Project Manager (80%) Arquitecto (20%)	425.37 euros

Figura 4. Tabla de costes humanos

Se han asignado a cada tarea los porcentajes que cada rol ejecutará en cada una de estas para así calcular mejor el coste. En la tarea de reuniones se necesita el mismo tiempo tanto al arquitecto como al Project manager por lo que ambos cuestan el 100% de la duración de la tarea.

Teniendo en cuenta que el coste bruto de recursos humanos es de 17339 y a continuación aplicamos el coste de seguridad social (multiplicamos por 1.35). El coste final de los recursos humanos es de 23407 euros.

6.1.2 Recursos materiales

Dentro de los recursos no humanos encontramos todos esos materiales necesarios para llevar a cabo el proyecto. Entre estos podemos encontrar ordenadores, licencias de programas o trasporte entre otros.

Lo primero que vamos a tener en cuenta son los cuatro portátiles que hemos utilizado en el proyecto. Está claro que no podemos cobrar al cliente el coste total del portátil por lo que hemos de calcular la amortización de estos. Pondremos una vida útil de 4 años.

Lo mismo haremos con las pantallas, les daremos una vida útil de 3 años.

Para hacer le cálculo de la amortización primero hemos de conseguir el coste residual. Utilizaremos como ejemplo un portátil Lenovo de 1000.78 euros. Como hemos dicho la vida útil son 4 años. Así pues:

- 1000.78 euros / 4 años = 250.195 euros/año.
- 1000.78 euros – 250.195 euros = 750.585 euros.
- 750.585 euros/ 4 años = 187.64 euros/ año.
- 187.64 euros/año ÷ 12 meses => 15.63 euros/mes * 4,5 meses => 70.36

Material	Unidades	Tiempo de uso	Precio inicial	Coste final
Portátil [12]	4	4,5 meses	1000.78	281.47
Pantalla [13]	4	4,5 meses	219	73
Licencia de IntelliJ 2019 [14]	2	4,5 meses	299	224.25

Figura 5. Tabla de materiales

No tenemos en cuenta los costes de servidor para los entornos de preproducción y producción dado que el cliente ya dispone de servidores propios, por lo que no supone un coste adicional para ellos.

El resto de software utilizado en el proyecto es de distribución libre y por lo tanto su uso no tiene coste alguno.

6.1.3 Costes indirectos

Dentro de los costes indirectos encontramos todos aquellos costes necesarios para el correcto trabajo. Entre estos hemos considerado como más relevantes: el internet y el coste de los portátiles.

Material	Unidades	Tiempo	Coste	Total
Portátiles	4	846h	0.19 kW/h 0,138065 Euro/kWh	88.77 euros
Internet	1	4.5 meses	200 euros/mes	900 euros

Figura 6. Tabla de consumos

6.1.4 Contingencias e imprevistos

Consideramos como costes imprevistos todos aquellos que pueden surgir y entorpecer el curso del proyecto. Por lo tanto, hay que hacer una serie de cálculos que podemos ver en la siguiente tabla.

Costes directos => 23407 de recursos humanos + 578.72 materiales = 23985.72 euros.

Costes indirectos => 988.77 euros.

Costes totales = 24974.49 euros.

Hemos destinado un 15% del coste total para imprevistos por un valor de 3746.17 euros.

6.2 Control de gestión

Es necesario mantener un control de gestión, para así poder calcular con facilidad posibles desviaciones. Tal hecho nos podría servir para identificar la fuente de los problemas y a solucionarlos con mayor facilidad.

El mayor coste es de los recursos humanos. Es por eso por lo que es el más importante en cuanto al cálculo de desviaciones [15]. Para poder controlar de forma correcta las desviaciones necesitamos saber las horas reales que cada persona implicada en una tarea le dedica a esta. Es por eso por lo que, utilizando el software de gestión JIRA, cada implicado en la tarea apuntará en esta el tiempo real dedicado. Así pues, utilizaremos la siguiente formula teniendo en cuenta los datos anteriormente mencionados:

Desviación en precio: (Coste estimado – Coste Real) / Horas reales.

Desviación en consumo: (Horas estimadas – Horas reales) * Coste estimado

Desviación total: (Coste estimado – Coste Real) / Horas reales + (Horas estimadas – horas reales) * Coste estimado.

En cuanto a la desviación de costes materiales, no hay una gran posible desviación dado que hemos tenido en cuenta las amortizaciones y averías en el presupuesto. Además, las cuotas de las licencias son anuales y la duración del proyecto es inferior a un año por lo que no variará.

En cuanto a costes indirectos, calcularemos directamente el total con la fórmula: Costes indirectos totales = Costes indirectos estimados – Costes indirectos reales.

7 Especificación del proyecto

La especificación de un proyecto de software consiste en la creación de un documento que recopila todas las necesidades de las partes interesadas. Se lleva a cabo a partir de varias reuniones donde participan los Stakeholders y la empresa que va a desarrollar el producto. Lo más común es que los desarrolladores no formen parte de este proceso pues no se entra en detalle sobre cómo se van a llevar a cabo las funcionalidades sino más bien cuales. En la especificación dividimos la toma de requisitos entre los funcionales y no funcionales.

Dado que ya hemos visto cuales son nuestros objetivos principales y específicos (requisitos funcionales) así como los requisitos no funcionales en el apartado 3, en el siguiente subapartado vamos a ver directamente las funcionalidades que se van a desarrollar. Dichas funcionalidades han sido acordadas bajo un presupuesto después de la toma de requisitos con la entidad.

7.1 Funcionalidades de la aplicación

Las funcionalidades de la aplicación se van a estructurar en 3 principales categorías, de forma que sea más sencillo identificarlas.

La primera categoría de funcionalidades será la obtención de datos. La obtención de datos se basará en obtener los datos de las fuentes diferentes mediante una serie de tecnologías. Obviamente, esta funcionalidad requerirá de un estudio intensivo de estas fuentes de datos, así como grandes configuraciones de estas tecnologías. Cabe resaltar que es extremadamente importante que esta funcionalidad sea altamente escalable. De esta forma, en el caso de que un futuro se quiera añadir una nueva fuente de origen se pueda hacer de forma sencilla y eficaz. Un error en el diseño de esta funcionalidad podría suponer una enorme desviación del proyecto en el caso de inserción de un nuevo método. Este tipo de errores conllevan una gran actualización del código y rediseño de la arquitectura de este para incorporar una nueva funcionalidad, lo que conlleva una gran cantidad de tiempo y dinero.

La segunda categoría de funcionalidades será la indexación de documentos. Esta categoría contendrá esas tareas que estén relacionadas con coger esos documentos obtenidos de diferentes puntos de origen y guardarlos de forma estructurada y eficiente, garantizando que luego su consumo sea óptimo. Esta fase tiene gran importancia y aquí reside toda la parte de diseño de modelo de datos. Es por esto, que habrá que desarrollar un programa JAVA para insertar estos datos en Solr, así como configurar Solr para recibirlos y guardarlos de la forma correcta.

Por último, encontramos la categoría de funcionalidades será la retribución de los datos. La retribución de los datos englobará todas esas funcionalidades que tendrán que ver con el consumo de nuestra API [16]. Primeramente, hay que tener claro cómo vamos a gestionar todas las peticiones a nuestro sistema. Es muy importante crear esta capa de abstracción, dado que Solr no está hecho con ese propósito y siempre es recomendable delegar ese trabajo a tecnologías hechas explícitamente para eso, en nuestro caso Nginx. Más adelante explicaremos como funciona y como vamos a utilizarla nosotros. Una vez hemos conseguido esa capa de abstracción tocará configurar Solr para que devuelva esos datos que nosotros hemos indexado previamente según las necesidades y lógica de negocio de la entidad para la cual desarrollamos el producto. Uno de los motivos por los que hemos escogido Solr como tecnología principal es por el hecho de que es altamente configurable y eso nos permite una retribución de datos personalizada. Además, hay una serie de funcionalidades muy interesantes como la detección del lenguaje de documentos que aportan valor al sistema y lo llevan un paso más allá de ser un simple buscador.

7.2 Índices

Los índices, son las diferentes categorías de documentos que tendremos en nuestra base de datos. Es decir, no vamos a estructurar toda nuestra información de forma simple, sino que crearemos diferentes categorías que nos permitirán guardar campos diferentes para cada una de ellas. Más allá de que información guardamos en cada categoría, también podremos personalizar diferentes configuraciones que nos ayudarán a luego obtener esta información.

Los índices pues que vamos a tratar son:

- Alumni
- Centros y cátedras
- Blogs
- Programas
- Prestamos
- MBA clubs
- Historias
- Sugerencias
- Historias
- Eventos

7.3 Puntos de orígen

Como se ha comentado anteriormente, en este proyecto recogemos información de diferentes origines de datos. El hecho de que la información esté repartida es uno de los motivos principales por el cual se ha llevado a cabo este proyecto. Así pues, vamos a ver cuáles son dichos origines de datos. Algo importante es el reaprovechamiento de código, por lo cual vamos a ver cuáles de los índices definidos en el apartado anterior tienen un origen de datos similar. Agrupar estos índices nos ahorrará tiempo y creará una aplicación más robusta y eficiente.

El primer grupo que encontramos está formado por los índices de Alumni, Blogs y Clubs MBA. El punto de origen de los datos de todos estos índices son

Urls que llevan a páginas HTML. El contenido sale de la propia página HTML, por lo que habrá que llevar a cabo algún tipo de procesado.

El segundo grupo que encontramos es el formado por Centros y Cátedras, Prestamos, Profesores y Sugerencias. El origen de los datos de estos índices es sencillo y cómodo. Los datos provienen de un archivo JSON los datos del cual se irán modificando conforme pase el tiempo y se actualicen. El procesado será simple pues java tiene una integración muy fuerte con JSON.

El siguiente grupo que encontramos es algo peculiar. Formado por Programas e Historias, es una combinación de los dos orígenes anteriores. Estos datos están incrustados en páginas HTML, pero como un JSON+LD [17]. Más adelante veremos su procesado. Cabe resaltar que, además, las Urls de las historias las obtendremos a través de RSS [18] con una paginación dado que hay una cantidad muy grande y que crece con el tiempo,

El último índice por tratar sería eventos. El índice de eventos es algo especial. Los eventos se crean, pasan y no vuelven a suceder. Es por eso por lo que su contenido no cambia. Los eventos se crean a través de una plataforma del cliente. Solr permite la indexación de documentos de uno en uno, a través de una llamada POST. Así pues, la indexación, modificación y borrado de eventos se hace llamando directamente a la API de Solr, y no requiere ningún procesado de datos por nuestra parte.

7.4 Frequencia de indexación

El proceso de indexación estará completamente automatizado. Es por eso por lo que hemos de definir para cada índice en el que indexamos documentos, cada cuanto tiempo indexaremos contenido. Esta frecuencia se podrá alterar manualmente a petición de la entidad. Además, a pesar de estar automatizado también cabrá la posibilidad de formar una indexación de documentos en cualquiera de los índices (o en todos), en el caso de haber una urgencia como podría ser la indexación de un evento de última hora. Las frecuencias de indexación serán:

Para los índices de Alumni, Centros y Cátedras, Programas, Préstamos, MBA Clubs, Publicaciones la indexación de contenido se llevará a cabo cada día de la semana, a las 10 y a las 14 horas.

Para el índice de Blogs, la indexación de contenido será cada media hora, pues está claro que es una categoría que en cualquier momento puede recibir un post, y necesita visibilidad inmediata dada la naturaleza de su contenido.

7.5 Diagrama de casos de uso

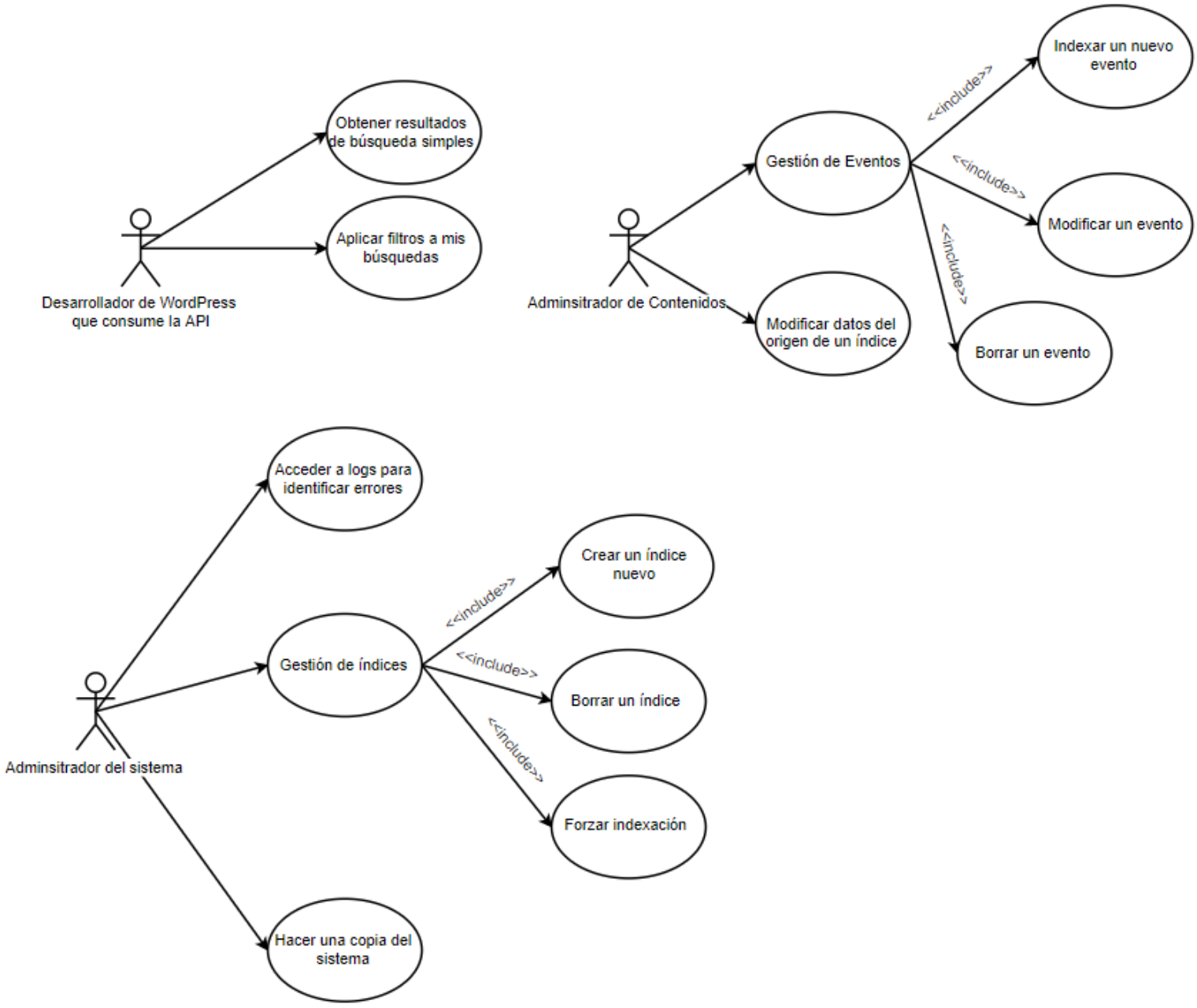

Figura 7. Diagrama de casos de uso

En un diagrama de casos de uso [19] podemos identificar todas las acciones posibles a realizar en un sistema. Encontramos dos tipos de figuras: las elipses y el icono de persona.

Los iconos de persona simplemente representan los denominados como Actores. Los actores son aquellas personas que ejecutan esas acciones. Por consecuente, las elipses son aquellas acciones llevadas a cabo por los actores.

Vamos a hablar sobre los 3 actores y que funcionalidades podrá ejecutar cada uno:

- Desarrollador de WordPress que consume la API: El desarrollador de wordpress solo tiene dos funcionalidades que utilizar que son busquedas tanto simples como con filtros. El será el que consuma la API y muestre visualmente los resultados obtenidos en Solr.
- Administrador de Contenidos: Observamos que tiene dos funcionalidades, la primera es modificar los datos del origen de un índice, mientras que la segunda es la gestion de eventos. La gestión de eventos es una funcionalidad compleja por lo que incluye diversas funcionalidades dentro de esta. Tanto indexar un evento nuevo, modificar uno existente y borrarlo, serán sub funcionalidades de esta y podrá ejectuarlas todas.
- Administrador del sistema: Es el actor que más funcionalidades utilizará. La primera es poder acceder a los logs para detectar errores que pueda haber en las indexaciones y detectar que lado, si el cliente o nuestra aplicación, es el culpable. La gestión de índices también es una función compleja e incluye: crear un índice, borrarlo y forzar su indexación. Más adelante veremos que para modificar un índice necesitaremos hacer una combinación de las 3. Por último podrá ejecutar copias de segurar del sistema de forma simple.

8 Diseño del proyecto

El diseño de un proyecto de software consiste en coger todos esos requisitos y funcionalidades del apartado anterior y responder a la pregunta de ¿Cómo lo vamos a llevar a cabo? Aquí es donde hay que tomar las decisiones más importantes como definir la arquitectura, el modelo de datos y las tecnologías a utilizar. Muchas veces entran en juego diagramas de flujo para definir el comportamiento de las funcionalidades, así como diagramas UML para definir los modelos de datos relacionales. En nuestro caso, utilizamos un modelo de datos no relacional (también conocido como NoSQL).

8.1 Tecnologías

Es importante meditar las tecnologías que escogemos y el porqué de estas, así pues, hay que conocerlas muy bien. Por eso, en los siguientes subapartados repasaremos una por una aquellas tecnologías y explicaremos cómo funcionan estas, de forma genérica, antes de explicar la implementación concreta que nosotros hemos hecho utilizándolas para solventar el problema de nuestros clientes.

8.1.1 Solr

Solr [20] es la tecnología principal de este proyecto y será nuestra base de datos. Sin embargo, Solr no nace de la nada. Esta es una implementación de Apache Lucene. Aunque más adelante veremos Solr en profundidad no está de más explicar un poco el funcionamiento de Lucene.

8.1.1.1 Apache Lucene

Lucene [21] es un proyecto de software libre de la gran compañía Apache. Este proyecto nació en 1997 y fue creado por *Doug Cutting*. Apache adquirió Lucene en el 2001 y en el 2005 pasó a ser un proyecto de importancia para dicha compañía. La importancia de este proyecto viene dada por el hecho de que corre bajo una licencia de software libre, por lo que cualquiera puede utilizar y contribuir a su desarrollo. Pero ¿qué es Apache Lucene?

Bien, apache Lucene no deja de ser una biblioteca de software. Lucene, base de datos no relacional es conocida por soportar inmensas cantidades de datos.

Pero lo que hace Lucene realmente increíble es su capacidad para buscar de forma muy eficiente. Este subdivide los documentos en campos de texto, por lo que el formato no tiene ningún tipo de importancia siempre y cuando sea texto. El uso de texto en vez de archivos convierte a Lucene una de las bibliotecas Java más rápida del mercado y recordemos, todo de forma completamente libre.

Lucene se creó para ser utilizado en JAVA, pero, sin embargo, tiene alternativas para .NET o C. Otras características que hacen destacar a Apache Lucene es su optimización de la memoria RAM o la velocidad de indexación (150GB por hora). Se puede buscar por campos de texto determinados o bien buscar por todos a su vez.

8.1.1.2 Apache Solr

Una vez visto cómo funciona Lucene, podemos adentrarnos en el funcionamiento de Solr en sí y así poder ver por qué escogimos esta tecnología con más profundidad de la vista hasta el momento.

Solr es una implementación de Lucene, es decir, podríamos interpretarlo como una capa por encima de este: tiene las mismas funcionalidades, pero añadiendo algunas nuevas.

¿Qué es lo que hace a Solr tan interesante pues? La respuesta es muy sencilla, el uso de índices invertidos de Lucene.

Los indicies invertidos son conocidos por su gran velocidad de búsqueda. Tienen un coste más elevado de preparación, pero dan resultados de forma espectacular. El funcionamiento de un índice invertido es muy simple. Explicaremos este funcionamiento con un ejemplo para verlo más claro. Supongamos que tenemos 100 archivos indexados y de los cuales solo 30 contienen la palabra perro. Un índice convencional iría archivo por archivo buscando la palabra perro y si la encuentra, añadiría dicho archivo a la lista de resultados. Un índice invertido hace todo lo contrario. En el momento de la

indexación del documento va a separar todas las palabras de ese documento y las guardará. Además, guardará una referencia a ese documento en la estructura donde guarda esas palabras. Por lo tanto, al buscar perro, iremos directamente a buscar entre nuestras palabras, buscaremos perro y devolveremos todas los archivos que tengamos relacionados a dicha palabra. Como ya hemos comentado, Lucene indexa texto, no archivos, por lo que el uso de índices invertidos se convierte en algo inmensamente efectivo y que solventa muchos problemas de eficiencia.

Aparte de esta gran arquitectura, vamos a repasar algunas de las funciones más importantes que aporta Solr a Lucene, y por lo tanto lo mejora:

- Uso de una caché para consultas recurrentes.
- Búsqueda teniendo en cuenta errores ortográficos.
- Clasificación de facetas.
- Búsqueda con comodín: Existe la posibilidad de buscar *erro o perr* y encontrará perro.
- Reconocimiento de texto de archivos Microsoft Word o PDF.
- Detector de idiomas.
- Integración directa con Java.

Todas estas funcionalidades aportan muchísimo valor a nuestra aplicación y además resuelven varios de los problemas planteados por la toma de requisitos. Así pues, podemos reafirmar que la elección de Solr sigue siendo la mejor y nos proporcionará una gran estabilidad y eficiencia en nuestra indexación y búsquedas.

8.1.2 Nginx y Openresty

Ya hemos comentado anteriormente que vamos a utilizar Nginx [22] como nuestro servidor Http. Es importante mantener una capa por delante de nuestra aplicación que se dedique a recibir las posibles llamadas a esta. Solr de por sí, ya levanta una instancia de Jetty, que es un contenedor de servlets y servidor http de eclipse. Sin embargo, al utilizar Nginx, nos aseguramos de tener una herramienta completamente específica para ese trabajo y aparte de que

estamos quitando trabajo a Jetty, también añadimos una capa de seguridad extra.

Pero no solo hemos escogido Nginx por esta razón, sino que además tiene una serie de funcionalidades de las que nos podemos beneficiar.

Nginx es Open Source, cosa que como hemos visto con anterioridad nos interesa y tiene una comunidad muy grande. Hay muchas empresas importantes que utilizan Nginx como su servidor Http como por ejemplo las plataformas de streaming Netflix y SoundCloud, así como una empresa tecnológica tan importante como GitHub.

Este servidor tiene la capacidad de gestionar cientos de llamadas simultaneas, por lo que podría salvar a la máquina de un ataque masivo de peticiones, cosa que mantendría el servidor caído durante un período de tiempo.

Además, Nginx tiene su propia caché de contenido que aporta una optimización a las llamadas a la API, así como una gran variedad de plugin para complementar sus usos.

Una de las características más interesantes es la capacidad para duplicar peticiones. Gracias a esta funcionalidad, si en un futuro quisiéramos crear un índice con el contenido de otros solo tendríamos que duplicar el tráfico de cada índice hacia el común. Se puede duplicar cualquier llamada http incluyendo el cuerpo de la petición en caso de que lo contenga.

Una vez visto que es lo que nos ofrece Nginx, introducimos OpenResty [23]. OpenResty no es más que una integración de Nginx, con un compilador del lenguaje de scripting Lua. Gracias a esto, vamos a poder modificar las peticiones entrantes y adaptarlas a lo que Solr nos pide. En la fase de implementación veremos muchas personalizaciones en cuanto a las llamadas que entran a Nginx y las que enviamos a Solr. Con Lua podemos gestionar estas personalizaciones de forma muy sencilla e incluso añadir alguna otra medida de seguridad.

Toda esta cantidad de funcionalidades y la fiabilidad expuesta por muchas empresas hace que nos decantemos por OpenResty como nuestro servidor Http.

8.1.3 Indexador Java, Crontab y Scripts

Después de la toma de requisitos hemos llegado a la conclusión de que nos hace falta algo muy importante: que los datos indexados se mantengan actualizados y que le proceso sea automático.

Una de las decisiones de utilizar Solr, era por su gran integración con el lenguaje de programación Java gracias a su librería SolrJ. Gracias a java vamos a poder crear un ejecutable, que cuando sea activado coja los datos del origen de cada índice, procese la información y la indexe en Solr.

Es una solución potente, robusta y eficaz. Gracias a la herramienta de construcción Maven, vamos a crear un JAR, es decir, el ejecutable. Dicho JAR estará subido en el servidor y contendrá nuestra aplicación de indexación. Pero este JAR tiene que ser ejecutado de alguna forma.

La forma más sencilla, es entrar en el servidor y ejecutar el JAR por consola cada vez que nos sea necesario para una indexación. Sin embargo, es insostenible. Nuestra solución es sencilla. Se van a crear diversos Scripts, escritos en bash dado que nuestro servidor es Linux. Una vez tengamos escrito un script para la indexación de cada índice, los subiremos al servidor y vamos a configurar una herramienta llamada Crontab [24].

Crontab no es más que lo que se conoce como un horario. Puedes configurarlo para que haga una acción determinada en una frecuencia determinada. Por ejemplo, podemos decir que ejecute el script de historias cada 30 minutos, sin embargo, que todos los otros sean a las 10 y a las 14 de cada día de la semana. De esta forma, si hay algún tipo de cambio, se podría realizar fácilmente y el acoplamiento entre índices es mínimo. Además, en el caso de necesitar forzar una indexación de uno o varios índices en un momento determinado, nos bastaría con ejecutar aquellos scripts que nos sean necesarios.

La información siempre se va a mantener actualizada pues cada vez que se ejecute nuestro Crontab, se irá a buscar la información al origen, y si se ve que ha habido alguna modificación, se actualizará, siendo un proceso eficiente y sencillo.

8.2 Arquitectura

Cuando hablamos de arquitectura, podemos referirnos a varias cosas. Lo primero es la infraestructura, cuantos servidores vamos a utilizar, cuantos procesadores, tamaño de memoria RAM y demás propiedades físicas (o desde hace ya un tiempo pueden ser virtuales).

Segundo, podríamos hablar de las tecnologías utilizadas y el flujo que toman los datos a lo largo de toda la aplicación.

Y, por último, encontramos la arquitectura del código, que se basa en como estructurarlo para que cumpla una serie de principios de calidad. Este último no lo vamos a tratar hasta la fase de implementación.

Por lo que en este apartado nos centraremos tanto en la infraestructura y distribución de servidores como el flujo de los datos desde su punto de origen hasta que son consumidos por el front-end de nuestra aplicación.

8.2.1 Servidores y flujo de datos

En todos los proyectos de desarrollo de software vamos a encontrar por lo menos dos entornos, el local y el de producción. Sin embargo, cada empresa añade en medio de estos dos la cantidad que crea y sea necesaria para cumplir sus requisitos.

Lo más común es añadir un entorno de preproducción para poder probar los desarrollos sin exponerlos al entorno de producción. Aparte, este entorno puede servir para que el cliente valide si la funcionalidad cumple los requisitos que debe y así dar su visto bueno para la salida a producción.

A partir de aquí, muchas empresas añaden un entorno de desarrollo, donde los desarrolladores pueden subir contenido y hacer todas las pruebas que deseen sin ningún tipo de peligro, pues el cliente no suele tener acceso a este entorno.

En nuestro caso, vamos a tener el entorno de producción, el de preproducción y el local.

Vamos a ayudarnos de los diagramas siguientes para poder explicar con mayor facilidad como está montada la infraestructura de cada uno de los entornos. Hay que resaltar que el entorno local simplemente utiliza instancias de esas tecnologías necesarias, replicando un servidor, pero que no permite acceso a otras máquinas a este, o por lo menos no en nuestro caso.

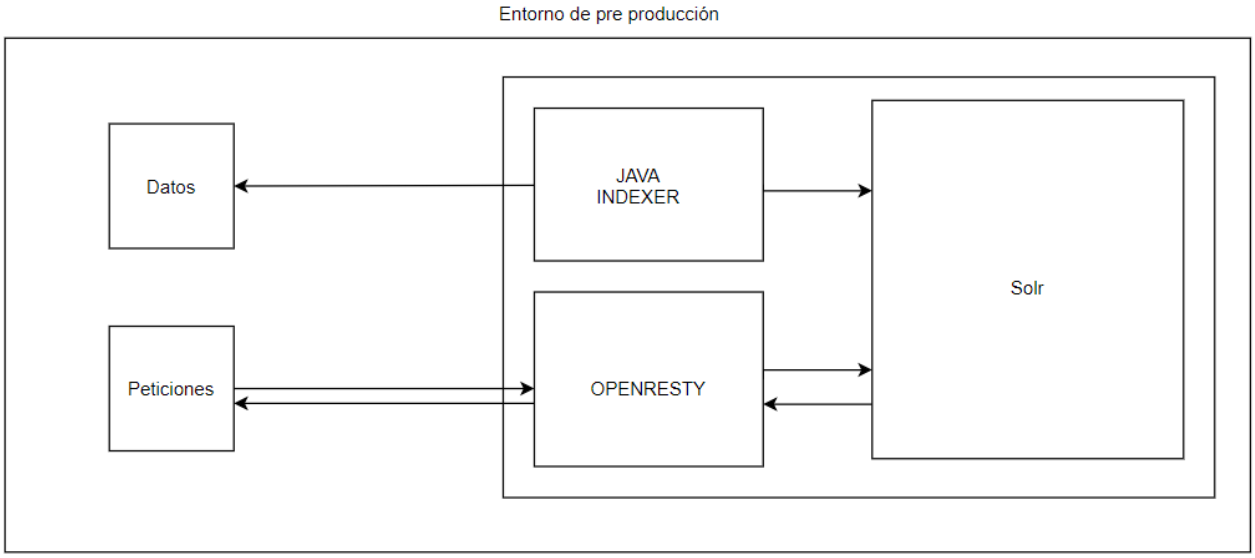

Figura 8. Arquitectura máquina pre producción

Bien, como podemos observar tenemos diferentes módulos. Por un lado, tenemos la recogida de datos. Esto se efectúa desde nuestro indexador JAVA. Vemos como es este el que va a buscar los datos. Una vez procesados correctamente según los algoritmos definidos, los insertamos dentro de Solr.

Paralelamente encontramos un segundo proceso, se ejecutan peticiones que como bien hemos explicado en apartados anteriores serán gestionadas por nuestro servidor http OpenResty. Este redirigirá las peticiones validas, filtrando

las invalidas, hacia Solr. Solr dará una respuesta que enviará a OpenResty y este la enviará de vuelta a la entidad que haya hecho dicha petición. Podemos observar que en el apartado de preproducción tenemos absolutamente todos los módulos en la misma máquina. Esto se debe a que el volumen de peticiones no es demasiado elevado, así como el de datos también es reducido y podemos tenerlo todo junto.

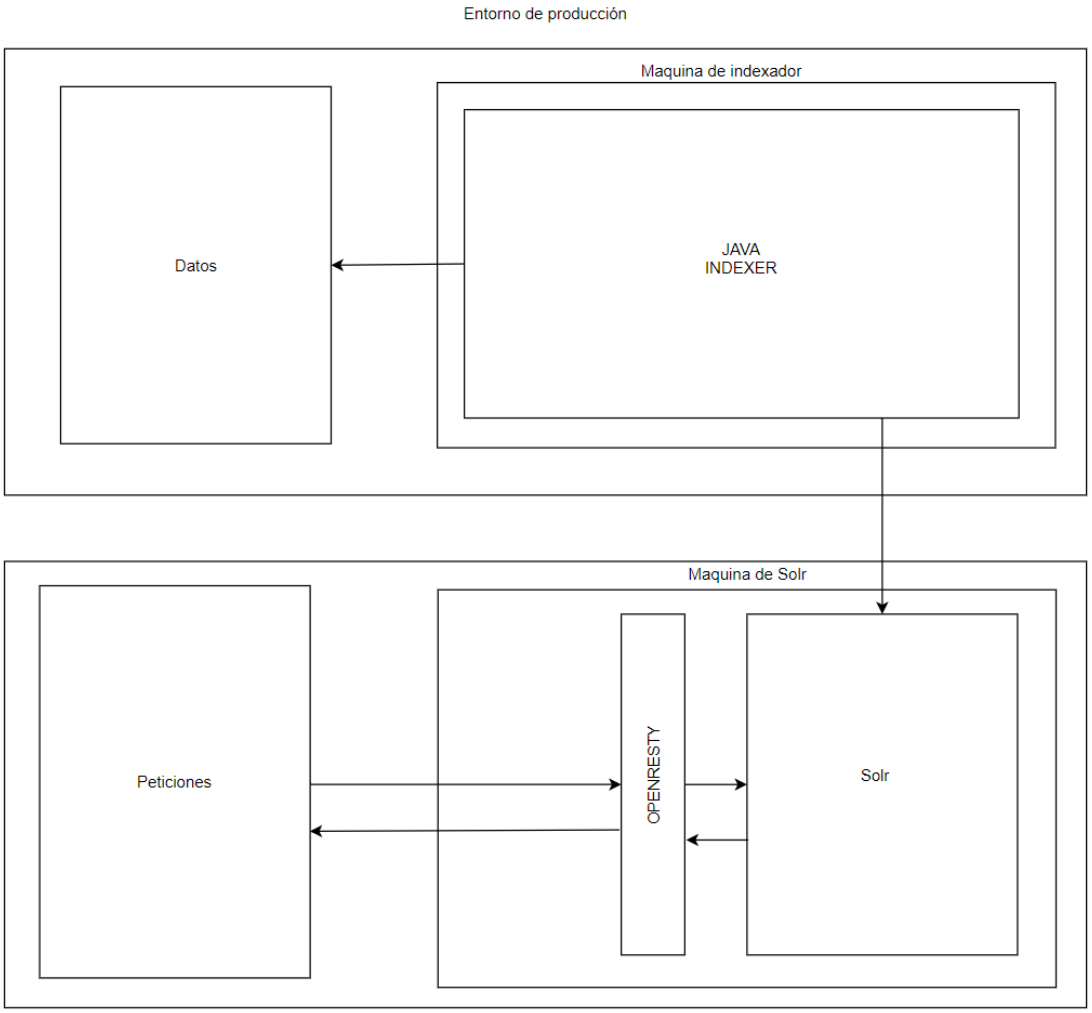

Figura 9. Arquitectura maquinas producción

Sin embargo, si miramos el diagrama anterior (figura 9), podemos observar como el flujo es algo distinto. Por supuesto, el indexador sigue recogiendo los datos de los puntos de origen pertinentes. Esta vez, la instancia de Solr se encontrará en una máquina independiente. Este recibirá los datos del

indexador y los guardará. Por otro lado, la instancia de OpenResty también se encuentra en la misma máquina de Solr, dado que esta recibirá las peticiones y las redirigirá a Solr, contestando más tarde al igual que en el entorno de preproducción.

La decisión de separar el indexador y Solar junto con OpenResty en dos maquina completamente distintas tiene una simple lógica. En el entorno de producción la cantidad de peticiones obtenidas es muy elevada, el servidor necesita la capacidad suficiente para poder procesar estás peticiones y a su vez, los datos obtenidos de los orígenes y guardados en Solr son grandes. Por eso, el indexador necesita una máquina aparte para no sobrecargar la maquina donde está Solr contenido. Aparte, en el caso de que le indexador tuviera algún tipo de error que hiciera que la máquina cayera, la única afectación que tendría Solr sería la desactualización de datos, pero podría seguir recibiendo peticiones y dando resultados.

8.3 Definición de llamadas de consumo a la API

Es importante definir que llamadas hay que construir en la aplicación para que la aplicación se construya entorno a ellas. Eso sí, hay que desacoplar completamente estas de los procesos de indexación para que este sea lo más escalable posible. Añadir una nueva llamada, debería conllevar más tiempo de configuración de cara a temas de seguridad que no de desarrollo en el caso de tener una aplicación desacoplada y construida eficientemente.

Una vez más, dividiremos estas llamadas según los índices que tengamos. Cada índice contendrá sus propios parámetros de búsqueda y filtros por lo que no tendría sentido hacer llamadas muy genéricas y tener que enviar campos vacíos. Todas las llamadas a la API serán a través del protocolo HTTP, dado que Solr expone los datos en una API REST [25]. Dicho esto, las llamadas son las siguientes:

Llamada de búsqueda	
método HTTP	GET
url	/search/{índice}
Descripción	Ruta de búsqueda para todos los índices
Parámetros query	
q	Contenido a buscar E.g: mba
format	Formato de la respuesta. Predeterminado: json
result_num	Numero de resultados. Predeterminado: 10
result_start	Número de resultado donde empezar. Predeterminado 0
filter	Filtrado de campos. E.g: language:es

Figura 10. Llamadas de búsqueda

Esta llamada nos permite hacer búsquedas en cualquier índice. Aunque estos son los parámetros que se están dejando configurar al usuario, cada índice tiene sus propios parámetros. Estos parámetros se añaden en Nginx cuando se redirige la llamada hacia Solr. Son siempre los mismos en cada índice, pero cada uno tiene los suyos propios. En el apartado de implementación de Nginx, podremos ver ejemplos de estos y toda la lógica que se ejecuta en este.

Llamada de eventos por venir	
método HTTP	GET
url	/search/nextEvents
Descripción	Devuelve los eventos a partir de la fecha de hoy
Parámetros query	
q	Contenido a buscar E.g: mba
format	Formato de la respuesta. Predeterminado: json
result_num	Numero de resultados. Predeterminado: 10
result_start	Número de resutlado donde empezar. Predeterminado 0
filter	Filtrado de campos. E.g: language:es

Figura 11. Llamada creación próximos eventos

Como respuesta de las llamadas de búsqueda solo hay 2 opciones, o bien un JSON vacío si no encuentra ningún resultado, o bien un JSON con resultados conteniendo los resultados. En el caso de que uno de los parámetros sea incorrecto se ignorará y si no se añade parámetro q, se devolverán todos los resultados (respetando los parámetros predeterminados como el número de resultados a 10).

Las siguientes llamadas son las que la empresa que controla el front-end hará sobre la API de Solr directamente sin nuestro procesado de datos. Aun así,

pasaran por nuestro Nginx y nosotros la redirigiremos por motivos de seguridad.

Crear evento	
método HTTP	POST
url	/index/events/create
Descripción	Llamada para crear un evento
Parámetros query	
Body	Json conteniendo los campos con los que se quiere crear el Índice

Figura 12. Llamada creación eventos

Modificar un evento	
método HTTP	POST
url	/index/events/update
Descripción	Llamada para modificar un evento
Parámetros query	
Body	Json conteniendo los campos que se quieren modificar del índice

Figura 13. Llamada actualización eventos

Borrar un evento	
método HTTP	POST
url	/index/events/delete
Descripción	Llamada para borrar un evento
Parámetros query	
Body	Json conteniendo los ids de los eventos a borrar

Figura 14. Llamada borrado eventos

9 Implementación del proyecto

9.1 Tareas a realizar

Las tareas de implementación están definidas en el apartado 5.1.4 Sin embargo, vamos a entrar más en detalle en cada una de ellas dado que es una de las fases más importantes del proyecto.

Definimos 5 tareas de implementación:

- TI1 preparación de entornos
- TI2 creación de índices
- TI3 Implementación del indexador,
- TI4 Habilitar búsquedas,
- TI5 Funcionalidades genéricas

En los apartados siguientes veremos en profundidad en qué consistirá cada una de las tareas de implementación. Añadiremos ejemplos de código y configuraciones para entender la nomenclatura de cada tecnología. Cabe destacar que es muy importante basarse en las fases de especificación y diseño durante el desarrollo, pues ahí es donde se ha definido toda la arquitectura y es necesario para el comportamiento óptimo y deseado de la aplicación.

9.1.1 TI1 Preparación de Entornos

Como hemos visto en la fase de diseño contamos con 1 servidor para pre producción, y otros dos para producción. Así pues, nos hará faltar preparar estos 3 servidores para poder trabajar con ellos. El más urgente es el de pre producción, pues es el que utilizaremos para hacer las pruebas, así como para que el cliente valide las funcionalidades a medida que las vayamos completando.

En cuanto a la preparación de entornos, hay una serie de pasos que hay que seguir. Siempre hay que tener muy en cuenta la seguridad. El primer paso sería instalar los certificados de SSL [26] para que se puedan hacer las peticiones a

través del protocolo HTTPs. Además, nos hará falta instalar todas aquellas dependencias necesarias para nuestras tecnologías, así como las propias tecnologías en sí. Así pues, las dependencias principales que deberemos instalar son:

- Java 1.8
- Maven 3.0
- OpenResty
- Solr 7.5
- Crontab

Nos conectaremos a estas maquinas a través del protocolo SSH por lo que deberemos generar una clave pública y una privada. La clave pública irá en el servidor, y la clave privada en la maquina local desde la cual querremos conectarnos al servidor. Estás claves tendrán asociadas un usuario y contraseña y así podremos conectarnos con algun programa como podría ser putty a los servidores, que tienen como sistema operativo Linux. Además, en el apartado de despliegues a producción, veremos que nos servirá tambien para conectarnos via SFTP y subir los archivos al servidor.

9.1.2 TI2 Creación de los índices de Solr

En esta tarea es donde se definen los indicies de Solr, por lo tanto, como se va a estructurar la información. Solr es un sistema de persistencia NoSQL [27], por lo que no lo podemos representar a través de un diagrama UML. Lo más sencillo será utilizar un Objeto JSON, que a final de cuentas es tal y como se representa en Solr.

Lo primero, es explicar de que está compuesto un índice. Principalmente, un índice está formado por dos fixeros XML, solrconfig.xml y schema.xml. Durante el análisis de estos índices, veremos que hay varios que comparten algunas funcionalidades y hay otros que destacan sobre los otros en cuanto a complejidad. Así pues, no tendría sentido explicar un solo índice, así como tampoco lo tendría explicarlos todos. De forma que lo que haremos será

agruparlos o bien id explicando las diferencias entre ellos gracias a ejemplos que ayudarán mucho a entender el funcionamiento e implementación de estos.

9.1.2.1 Definición de ficheros SolrConfig.xml

Solrconfig.xml o configuración del índice, es el responsable del comportamiento de los datos del índice. Aquí definimos muchas de las acciones que queremos que se ejecuten, así como la forma de tratar los documentos que indexamos. Está configuración está dividida en varias partes a través de un fichero XML.

```
<directoryFactory name="DirectoryFactory"
                  class="${solr.directoryFactory:solr.NRTCachingDirectoryFactory}"/>

<codecFactory class="solr.SchemaCodecFactory"/>
```

Figura 15. Factorías

Lo primero es definir las factorías. Hemos escogido las predeterminadas, que básicamente hacen que sea un sistema basado en ficheros, con uso de caché y que el códec oficial de Lucene, dado que el resto son experimentales.

Seguidamente encontramos la configuración de logs que ofrece Solr de forma implícita.

```
<updateHandler class="solr.DirectUpdateHandler2">

  <updateLog>
    <str name="dir">${solr.ulog.dir:}</str>
    <int name="numVersionBuckets">${solr.ulog.numVersionBuckets:65536}</int>
  </updateLog>

  <autoCommit>
    <maxTime>${solr.autoCommit.maxTime:15000}</maxTime>
    <openSearcher>false</openSearcher>
  </autoCommit>

  <autoSoftCommit>
    <maxTime>${solr.autoSoftCommit.maxTime:-1}</maxTime>
  </autoSoftCommit>

</updateHandler>
```

Figura 16. Configuraciones predeterminadas

Estas configuraciones son las predeterminadas y recomendadas por Apache en el uso de Solr, son las mismas para todos los índices.

El siguiente paso es la definición de cachés en el apartado de query. Nos servirá para optimizar mucho el funcionamiento de Solr. Además, definiremos el número de documentos a guardar en la cache.

```
<query>

  <filterCache class="solr.FastLRUCache"
               size="512"
               initialSize="512"
               autowarmCount="0"/>


  <queryResultCache class="solr.LRUCache"
                    size="512"
                    initialSize="512"
                    autowarmCount="0"/>



  <documentCache class="solr.LRUCache"
                 size="512"
                 initialSize="512"
                 autowarmCount="0"/>

  <cache name="perSegFilter"
         class="solr.search.LRUCache"
         size="10"
         initialSize="0"
         autowarmCount="10"
         regenerator="solr.NoOpRegenerator" />

  <queryResultMaxDocsCached>200</queryResultMaxDocsCached>

</query>
```

Figura 17. Cachés

Hasta aquí, todos los índices tienen las mismas propiedades. Sin embargo, a partir de ahora es cuando empezaremos a ver las claras diferencias. El último apartado de la configuración está destinado a los denominados request Handlers. Estos request handlers son los que definen como se comportan las búsquedas e indexaciones en el índice. La forma de tratar la información de los documentos, la forma de devolverlos incluso como actualizarlos. Algo muy interesante es el uso de componentes, los cuales vienen dados por Solr y nos permiten ejecutar acciones muy interesantes en algunas de nuestras peticiones.

La petición básica, es la de búsqueda.

```
<requestHandler name="/select" class="solr.SearchHandler">

  <lst name="defaults">

    <str name="echoParams">explicit</str>
    <int name="rows">10</int>

  </lst>
</requestHandler>
```

Figura 18. Petición de búsqueda

Como podemos observar, es un Handler muy simple. Definimos la ruta con el nombre, en este caso name=" /select". También le asignamos la clase java que va a gestionar la petición. Cabe destacar que Solr da muchas clases para gestionar las peticiones, pero si ninguna cumple con los requisitos necesarios, se puede crear una propia. Los parámetros echoParams contiene los campos a devolver en el caso de que activemos la petición en modo de depuración. En este caso usamos "explicit" para que devuelva todos. El parámetro rows nos indica el número de resultados a devolver de forma predeterminada.

```
<lst name="appends">
  <str name="fq">inStock:true</str>
</lst>
```

Figura 18. Filtro

A partir de este Handler tan básico, podríamos complicarlo todo lo que quisiéramos. Uno de los más básicos sería aplicar un filtro por ejemplo diciendo que nos devuelva los resultados que tengan el campo inStock a verdadero.

Todos los índices excepto uno comparte el mismo Handler de "/select", sin embargo, encontramos el de profesores que es un poco especial y vamos a analizar a continuación.

```
<requestHandler name="/select" class="solr.SearchHandler">

  <lst name="defaults">
    <str name="echoParams">explicit</str>
    <int name="rows">10</int>

    <str name="q">*</str>
    <str name="defType">edismax</str>
    <str name="qf">phonetic_name^100 first_name^100 surname_1^100 surname_2 dedication_type departments email
interests chair_member chair_holder director_centers sector subsector cv_summary cv_complete twitter_user</str>

    <str name="bf">if(eq(professor_type,"professor"),300,1)</str>

  </lst>

</requestHandler>
```

Figura 19. Petición de búsqueda avanzada

Como podemos observar, vemos campos que no habíamos visto hasta ahora. En el tag del XML <lst name=" defaults"> ahora encontramos varios parámetros más aparte del echoParams y rows. El primero, la q nos indica que hemos de buscar, y este nos indica que hemos de buscar todo con un *. Esta búsqueda es la predeterminada, se puede sobrescribir poniendo una alternativa de forma implícita en la petición. ¿Por qué el resto de los índices no tienen esta definición predeterminada? Por el hecho de que ahora vamos a definir un defType. En el campo defType lo que estamos definiendo es lo denominado como Query Parser. Un query Parser es el componente encargado de transformar el texto de la petición al objecto de Lucene Correspondiente. Encontramos principalmente 3:

- Standard: Es el más simple de todos. Permite aplicar filtros simples y devuelve los resultados en un objecto results. [28]
- DisMax: Permite hacer peticiones a través de multiples campos con diferentes pesos. [29]
- eDisMax: Se comporta igual que el DisMax pero contiene mas funcionalidades. La “e” hace referencia a extended. [30]

Hemos elegido el Query Parser eDisMax por el hecho de que nos va a permitir hacer búsquedas por diferentes campos añadiendo pesos a aquellos campos que queramos dar más importancia. En nuestro caso, vamos a dar un peso de 100 veces más importante a los campos phonetic_name, first_name y

surname_1. Este Query Parser lo vamos a utilizar en la mayoría de los índices. Sin embargo, en muchos de ellos hemos de variar en cada request los campos en los que buscamos o hemos de hacer algún tipo de modificación. En el índice de profesores, la búsqueda siempre será la misma así que nos ahorramos añadir la selección de Query Parser y campos donde buscar en la petición y la añadimos directamente al Handler. Más adelante, cuando veamos los ejemplos de peticiones en OpenResty, se verá la lógica que siguen el resto de los índices.

Finalmente, encontramos el filtro bf que , en inglés, hace referencia a boost field. Con el boost field lo que hacemos es decir que en el caso de que el campo professor_type sea del tipo professor, multiplique por 300 el peso de este resultado.

Para la mayoría de los índices, solo vamos a usar el Handler “/select” porque solo necesitamos hacer consultas, dado que los datos se modifican gracias a la aplicación, y se borran o actualizan en el propio origen de los datos. El único índice que no pasa por nuestro indexador Java es Eventos, por lo que sí que vamos a implementar los handlers de crear, actualizar y borrar.

```
<requestHandler name="/create/json" class="solr.UpdateRequestHandler">
      <lst name="defaults">
       <str name="stream.contentType">application/json</str>
     </lst>
</requestHandler>

<requestHandler name="/update/json" class="solr.UpdateRequestHandler">
      <lst name="defaults">
       <str name="stream.contentType">application/json</str>
     </lst>
</requestHandler>

<requestHandler name="/delete/json" class="solr.UpdateRequestHandler">
      <lst name="defaults">
       <str name="stream.contentType">application/json</str>
     </lst>
</requestHandler>
```

Figura 20. Peticiones eventos

Para Solr, estas tres acciones se interpretan como una actualización y es por eso por lo que podemos observar como la clase de Solr correspondiente, solr.UpdateRequestHandler, es la misma en las 3. Simplemente definimos las 3 rutas, e indicamos que le contentType del stream que vamos a recibir será json.

Para terminar, nos haría falta ver los componentes que se pueden añadir. Para justificar el uso de estos componentes, vamos a ver primero tanto la definición de Schemas.xml como ejemplos de peticiones, y en un apartado futuro veremos en qué consisten dichos componentes y el porqué de su uso.

9.1.2.2 Definición de ficheros Schema.xml

En los ficheros Schema.xml es donde definimos que campos van a contener cada índice. Se podría hacer una similitud con lo que sería una tabla en una base de datos relacional y sus atributos. Una clara diferencia, es que aquí, no tenemos por qué guardar siempre un resultado(ni siquiera vacío), es decir, podemos rellenar de un objeto simplemente 3 de 5 campos de su schema para que así no ocupen memoria. Dicho esto, pasemos a ver cuál es la estructura básica de un fichero schema.xml y como lo hemos adaptado nosotros para nuestros casos específicos.

Un schema, se divide principalmente en 3 partes, la primera son los tipos. Tenemos que definir los tipos, dado que es obligatorio asignarle un tipo a un campo. Diferentes campos pueden tener un mismo tipo, sin embargo, un campo solo puede tener un tipo. Solr nos proporciona una serie de tipos predeterminados, pero es cierto que una buena definición de tipos permite una personalización muy amplia del comportamiento de los campos y la eficiencia de la búsqueda.

```
<fieldType name="string" class="solr.StrField" sortMissingLast="true" omitNorms="true"/>

<fieldtype name="binary" class="solr.BinaryField"/>

<fieldType name="int" class="solr.TrieIntField" precisionStep="0" omitNorms="true" positionIncrementGap="0"/>

<fieldType name="float" class="solr.TrieFloatField" precisionStep="0" omitNorms="true"
positionIncrementGap="0"/>

<fieldType name="long" class="solr.TrieLongField" precisionStep="0" omitNorms="true" positionIncrementGap="0"/>

<fieldType name="double" class="solr.TrieDoubleField" precisionStep="0" omitNorms="true"
positionIncrementGap="0"/>

<fieldType name="tint" class="solr.TrieIntField" precisionStep="8" omitNorms="true" positionIncrementGap="0"/>

<fieldType name="tfloat" class="solr.TrieFloatField" precisionStep="8" omitNorms="true"
positionIncrementGap="0"/>

<fieldType name="tlong" class="solr.TrieLongField" precisionStep="8" omitNorms="true"
positionIncrementGap="0"/>

<fieldType name="tdouble" class="solr.TrieDoubleField" precisionStep="8" omitNorms="true"
positionIncrementGap="0"/>

<fieldType name="tints" class="solr.TrieIntField" docValues="true" precisionStep="8" positionIncrementGap="0"
multiValued="true"/>

<fieldType name="tfloats" class="solr.TrieFloatField" docValues="true" precisionStep="8"
positionIncrementGap="0" multiValued="true"/>

<fieldType name="tlongs" class="solr.TrieLongField" docValues="true" precisionStep="8" positionIncrementGap="0"
multiValued="true"/>

<fieldType name="tdoubles" class="solr.TrieDoubleField" docValues="true" precisionStep="8"
positionIncrementGap="0" multiValued="true"/>

<fieldType name="date" class="solr.TrieDateField" omitNorms="true" precisionStep="0" positionIncrementGap="0"/>

<fieldType name="location" class="solr.LatLonType" subFieldSuffix="_coordinate"/>

<fieldType name="tdate" class="solr.TrieDateField" omitNorms="true" precisionStep="6"
positionIncrementGap="0"/>

<fieldType name="tdates" class="solr.TrieDateField" docValues="true" precisionStep="6" positionIncrementGap="0"
multiValued="true"/>
```

Figura 21. Definición de tipos simples

Estos son algunos de los tipos simples que Solr nos ofrece de forma predeterminada. Vemos que tienen diferentes atributos. Lo primero es el nombre para poder identificarlo, y la clase que le corresponde, lo cual es básico. A partir de ahí, hay un montón de atributos más que podemos incluir. Al ser demasiados, se puede consultar su explicación en esta página de documentación.

Seguidamente, encontramos la parte realmente interesante de los tipos de campo. A un tipo, podemos añadirlo lo que se conoce como analyzers. Cuando nosotros introducimos datos en un campo, este analyzer es el encargado de tratar el texto recibido y convertirlo en un token. Este token es lo que realmente

se indexa en Solr y no el texto en sí, por lo tanto, lo que nosotros luego buscaremos. Con estos analyzers podemos dividir las palabras, eliminar espacios en blanco o aplicar filtros, lo cual nos permite personalizar mucho la indexación de documentos. Vamos a ver algún ejemplo de analyzers que hemos utilizado y el por qué.

```
<fieldType class="solr.TextField" name="text_general" positionIncrementGap="100">
  <analyzer type="index">
    <tokenizer class="solr.StandardTokenizerFactory"/>
    <filter class="solr.StopFilterFactory" ignoreCase="true" words="stopwords.txt"/>
    <filter class="solr.LowerCaseFilterFactory"/>
    <filter class="solr.ASCIIFoldingFilterFactory"/>
  </analyzer>
  <analyzer type="query">
    <tokenizer class="solr.StandardTokenizerFactory"/>
    <filter class="solr.StopFilterFactory" ignoreCase="true" words="stopwords.txt"/>
    <filter class="solr.SynonymGraphFilterFactory" expand="true" ignoreCase="true" synonyms="synonyms.txt"/>
    <filter class="solr.LowerCaseFilterFactory"/>
    <filter class="solr.ASCIIFoldingFilterFactory"/>
  </analyzer>
</fieldType>
```

Figura 22. Definición de tipo complejo

Aquí podemos observar el tipo de campo text_general. Es uno de los más utilizados a lo largo del proyecto por lo que vale la pena comentarlo. Lo primero que vemos es que podemos definir más de un analyzer en un tipo. Esto se debe porque el primero analizará el texto en el momento de indexar, y el segundo analizará los tokens indexados en el momento de buscar.

Lo segundo que observamos, es que aparece algo llamado tokenizer [31]. Hemos comentado el tema de los tokens, pues el tokenizer es el encargado de tratar esos campos y convertirlos en tokens. Hay una larga lista de tokenizer, pero en este caso vamos a utilizar el StandardTokenizer. Básicamente, crea tokens a partir de espacios en blanco y signos de puntuación. Sin embargo, los puntos que no son seguidos por un espacio se conservan en el token y la @ actúa como signo de puntuación. Por ejemplo, el texto “Hola, esto es un texto de test. Enviar email a test123@test.com”, resultaría en: “Hola” “esto” “es” “un” “texto” “de” “test” “Enviar” “email” “a” “test123” “test.com”.

Existen más tokenizers comunes como por ejemplo el Keyword Tokenizer que trata todo el texto como un solo token, el LowerCase Tokenizer que convierte

todo a minúscula o el Regular Expression Pattern Tokenizer que nos permite crear tokens a partir de expresiones regulares. Los tokenizers son una herramienta muy potente, y bien utilizada puede estructurar la información de una forma muy eficiente.

Seguidamente, encontramos los filtros [32]. Los filtros, tienen una función muy similar a los tokenizers. Estos repasan los tokens de una secuencia y deciden si los dejan pasar, si los modifican o si los quitan de la indexación. Hay muchísimos filtros, por lo que comentaremos únicamente los que se están utilizando en este tipo.

- StopFilterFactory. nos permite parar el analisis en unas palabras determinadas. Estas palabras están especificadas en un fichero de texto.
- LoweCaseFilterFactory: convierte todos los caracteres en mayúscula de un token a minúscula, sin alterar ningún otro carácter.
- ASCIIFoldingFilterFactory: en el caso de encontrar un carácter alfabético, numerico o simbolico que no esté en el Basic Latin Unicode Bloc, lo convierte a su equivalente ASCII, en el caso de que exista este.
- SynonymGraphFilterFactory: nos permite declarar una lista de sinonimos en un fichero de text. Esta lista contiene en cada linea, una palabra con sus equivalentes. Podemos fijarnos que este filtro solo está añadido en el analyzer de búsqueda dado que no nos interesa indexar todos esos sinonimos en nuestra base de datos, sino simplemente analizarlos cuando buscamos. Las listas de sinonimos son bidireccionales, es decir, se aplican para todas las palabras de una misma fila.

Gracias a este ejemplo hemos podido ver el funcionamiento de la mayoría de los tipos de campo complejos, simplemente haría falta hacer distintas combinaciones de tokenizers y filtros. Algo interesante a comentar es que, en la mayoría de los casos, si queremos obtener resultados de búsqueda siguiendo la misma lógica de tokenización, tendremos que utilizar los mismos filtros y tokenizers tanto en el analyzer de búsqueda como en el de

indexación. A pesar de esto, ya hemos visto que a veces nos sale rentable hacer pequeñas modificaciones como en el caso de los sinónimos.

Hay un último tipo de campo utilizado en este proyecto que vale la pena comentar dado que se diferencia sobre los demás, la búsqueda fonética. Los profesores de esta universidad son, en gran medida, internacionales. Eso significa que muchos de ellos tienen apellidos complejos que cuando hay que buscarlos son difíciles de recordar y escribir. Uno de los puntos por los que escogimos Solr fue por la posibilidad de añadir la búsqueda fonética que resolverá este problema con facilidad.

```
<fieldtype name="phonetic" stored="false" indexed="true" class="solr.TextField" >
  <analyzer type="index">
    <tokenizer class="solr.StandardTokenizerFactory"/>
          <filter class="solr.BeiderMorseFilterFactory" nameType="GENERIC" ruleType="APPROX" concat="true"
languageSet="auto"/>
  </analyzer>
  <analyzer type="query">
    <tokenizer class="solr.StandardTokenizerFactory"/>
    <filter class="solr.StopFilterFactory" ignoreCase="true" words="stopwords_names.txt"/>
          <filter class="solr.BeiderMorseFilterFactory" nameType="GENERIC" ruleType="APPROX" concat="true"
languageSet="auto"/>
  </analyzer>
</fieldtype>
```

Figura 23. Definición de filtro de búsqueda fonética

Para activar la búsqueda fonética es necesario crear un tipo de campo, que en un futuro aplicaremos a un campo. Ya conocemos lo que hace el StandardTokenizer así que pasaremos directos a lo nuevo, el filtro BeiderMorseFilterFactory. Este filtro aplica un algoritmo desarrollado únicamente para detectar nombres que son iguales, escritos de forma diferente o en diferentes idiomas. Además, podemos escoger entre si queremos que sea una búsqueda de nombres genérica o si queremos buscar nombres sefardíes o asquenazíes. En nuestro caso usaremos el genérico, pero ya vemos que esta herramienta soluciona con creces nuestro problema. Algo curioso, es que había apellidos que contenían la palabra de, y la búsqueda fonética generaba conflictos cuando alguien buscaba opus dei. De forma que fue necesario añadir un filtro de StopWords en la búsqueda con tal de que esa palabra no devolviera

profesores que no tuvieran relación con esta institución. En el caso de haber añadido este filtro en el momento de indexación, todos aquellos apellidos con la palabra de, la hubieran perdido.

Una vez visto esto, podemos pasar a ver como se construyen los campos en sí, dado que hasta ahora solo hemos visto los tipos de dichos campos. Una vez más, utilizaremos una serie de ejemplos para mostrar su funcionamiento.

```
<field name="id" type="string" stored="true" indexed="true" required="true"/>
<field name="_version_" type="long" indexed="true" stored="true"/>

<!-- core fields -->
<field name="segment" type="string" stored="true" indexed="false"/>
<field name="digest" type="string" stored="true" indexed="false"/>
<field name="boost" type="float" stored="true" indexed="false"/>

<!-- fields for index-basic plugin -->
<field name="host" type="url" stored="false" indexed="true"/>
<field name="url" type="url" stored="true" indexed="true"/>
<!-- stored=true for highlighting, use term vectors  and positions for fast highlighting -->
<field name="content" type="text_general" stored="true" indexed="true"/>
<field name="title" type="text_general" stored="true" indexed="true"/>
<field name="cache" type="string" stored="true" indexed="false"/>
<field name="tstamp" type="date" stored="true" indexed="false"/>
<field name="chair_member" type="text_general" stored="true" indexed="true" multiValued="true"/>
<field name="phonetic_name" type="phonetic" stored="true" indexed="true" multiValued="true"/>
```

Figura 24. Definición de campos

Los primeros campos que vamos a ver son los simples. Estos campos como vemos tienen un nombre y un tipo siempre. Además, observamos que los atributos stored e indexed siempre aparecen. El atributo stored activado nos indica que ese campo siempre se va a ver en la respuesta de una búsqueda y el campo indexed, que indexamos el texto de ese campo como tokens para poder hacer búsquedas sobre él. Así pues, por mucho que nunca busquemos en el campo tstamp, cuando hagamos una búsqueda sobre el campo id, por ejemplo, el json de respuesta contendrá para cada uno de sus resultados el campo tstamp si es que lo tiene. El atributo requiered, si está activo, nos indica que ese campo es obligatorio rellenarlo. Por último, en estos campos simples, podemos añadir el atributo

multiValued que convertirá ese campo en una lista, por lo que será un campo múltiple.

Obviamente, si asignamos a un campo el tipo phonetic, como es el caso del campo phonetic_name, habilitamos para ese campo la búsqueda fonética.

```
<copyField source="first_name" dest="phonetic_name"/>
<copyField source="real_name" dest="phonetic_name"/>
<copyField source="surname_1" dest="phonetic_name"/>
<copyField source="surname_2" dest="phonetic_name"/>

<dynamicField name="*_s" type="text_general" indexed="true"  stored="true"/>
```

Figura 25. Definición de copyFields

El siguiente campo que vamos a ver, es el denominado como copyField. Como su nombre indica, lo que hace es copiar contenido de un campo a otro. En este caso estamos copiando todo el contenido de first_name, real_name, surname_1 y surname_2 a un campo de destino phonetic_name que va a englobar todos ellos. De esta forma, podemos aplicar la búsqueda fonética sobre un solo campo, pero con el contenido de muchos otros. Los campos de origen y destino han de existir para el correcto funcionamiento de los copyFields.

Para finalizar con los campos del fichero schema.xml, vamos a describir los dynamicFields. Estos campos actúan como un comodín, si indexamos a través de la petición un campo que coincida con una expresión de un dynamicField, va a ser creado. En el ejemplo anterior vemos que existe un dynamicField llamado *_s, de forma que si hacemos una petición con los campos test1_s y test2_s indexaremos ambos sin necesidad de añadir los dos en el schema. Por supuesto, tendrán el mismo comportamiento dado que los atributos del dynamicField serán los mismos para ambos.

Con esto concluimos la explicación de la tarea de implementación 2 que era la creación de índices. Más adelante, en la tarea número 5, habilitaremos unas mejoras para estos índices con componentes y funcionalidades especiales.

9.1.3 TI3 Implementación del Indexador

La implementación del indexador se llevará a cabo con tecnologías como Maven y Java, gracias a la librería que proporciona apache, SolrJ. Con ella podremos recoger la información en los puntos de origen, tratar dicha información y enviarla a los distintos índices de Solr donde nos conectaremos para indexarla.

El indexador es una aplicación simple pero efectiva. Para ejecutarlo debemos invocar el siguiente comando:

Java -jar solr-indexer-XXX.jar -seed solr-indexer\src\scripts\content_pro\<indice>-seed.txt -core financial -server <server_url> -update -user <wordpress_user> -password <wordpress_pass> -solr-user <solr_user> -solr-password <solr_password>

Veamos los parámetros:

- Seed: Fichero que contiene el punto de origen de donde cogeremos los datos a indexar.
- Core: Índice que queremos actualizar.
- Server: url del servidor que queremos actualizar, puede ser local, pre producción o producción.
- Update: si no lo ponemos los cambios no se efectuaran, nos sirve para hacer pruebas y ver los logs informativos.
- User: El usuario de wordpress.
- Password: La contraseña de wordpress.
- Solr-user: El usuario de Solr.
- Solr-password: La contraseña de Solr.

Como ya sabemos tenemos varios puntos de origen y para cada uno de estos puntos de origen, un Parser que nos transformará los datos para que los podamos tratar correctamente.

Una vez que hemos conseguido extraer los datos con cada uno de los Parsers, crearemos lo que hemos denominado como bean. Un bean, no deja de ser un objecto para indexar en Solr. Este bean estará formado por atributos que serán Strings o List<String> en el caso de que sean multiValued. Además, estarán marcados por un tag tal que así:

```
@Field("author")
private List<String> authors;

@Field("source_blog_name")
private String sourceBlogName;
```

Figura 26. Definición de Fields en el Bean

Este nombre de @Field(“<nombre>”) ha de corresponder con el del nombre del campo del schema y el mapeo es directo. Una vez hemos guardado en una lista todos los objectos a indexar vamos a pedir a Solr todos los objectos que ya tiene. En el caso de que el objecto que vayamos a indexar esté ya en Solr, lo actualizaremos y lo borraremos de la lista de objetos que ya tenemos. En el caso de que no esté, simplemente lo indexaremos. Antes de finalizar, miraremos nuestra lista de Objetos que Solr ya tenía antes de empezar la indexación y si no está vacía significa que ha sido borrada del origen. De esta forma, esos objetos se borrarán de Solr.

A esta altura, el proceso habrá terminado y los datos de Solr habrán sido modificados con los nuevos datos del origen. Cabe resaltar la eficiencia del programa ya que añade, actualiza y borra en un mismo proceso gracias a la recuperación de datos de Solr tan eficiente que nos permite recorriendo una sola vez la lista de objetos actuales, llevar a cabo esta acción.

9.1.4 TI4 Habilitar búsquedas y filtros

Ya hemos creado nuestros índices, ya hemos indexado los datos en Solr, así que para que el front-end pueda consumir estos datos nos falta por supuesto habilitar las búsquedas y ya podremos empezar a ejecutar consultas. Como hemos comentado anteriormente, vamos a utilizar OpenResty como servidor http para habilitar las consultas.

Una vez más, tenemos muchos casos similares, así que vamos a mostrar distintos ejemplos de casos concretos para entender un poco como ha sido implementado el servidor OpenResty, así como las consultas a este.

Primero de todo vamos a definir unos parámetros comunes para todas las peticiones que tendremos.

```
if ($arg_format = '') {
    set $arg_format 'json';
}


if ($arg_result_num = '') {
    set $arg_result_num 10;
}

if ($arg_result_start = '') {
    set $arg_result_start 0;
}

if ($arg_q = '') {
    set $arg_q '*';
}


set $commonParams fq=$arg_filter&omitHeader=true&wt=$arg_format&rows=$arg_result_num&start=$arg_result_start;
```

Figura 27. Parámetros predeterminados de las peticiones

Creamos la variable $commonParams y gracias a $arg_* podemos obtener cualquier parámetro de la petición, como por ejemplo $arg_filter nos proporcionará el parámetro para filtrar. A estos parámetros comunes les añadimos un valor predeterminado en el caso de que vengan vacíos para no obtener ningún error.

Para definir una nueva ruta utilizaremos la palabra location seguida de la ruta que queremos definir. Abriremos corchetes y dentro podremos desarrollar toda la lógica para esa petición. Podemos encontrar peticiones muy sencillas como la del índice de Alumni, otras intermedias como el índice de Clubs Mba, y otras con mucha lógica aplicada incluso utilizando scripts en Lua como la de préstamos .

```
location /search/alumni-chapters/ {
    proxy_pass  $solr_server/alumni-chapters/select?q=title:$arg_q^100&$commonParams;
}
```

Figura 28. Petición del indice Alumni

Para dejar pasar una petición, utilizaremos la instrucción proxy_pass. Vemos que hay una variable $solrServer, que contiene la dirección del servidor correspondiente que obviamente no se muestra por un tema de confidencialidad. Seguidamente definimos la ruta la cual coincide con la generada en el Handler de Solr y a continuación los parámetros de query. La q nos índica cual es nuestra búsqueda. Recordemos que con $arg_q estamos cogiendo el parámetro q de la petición que nos ha llegado a dicha ruta por lo que estamos buscando este mismo parámetro q en el campo title y con una elevación de peso de 100. Además, añadimos los parámetros comunes que hemos definido antes. Una petición sencilla pero efectiva.

```
location /search/mba-clubs/ {
    if ($arg_q = '*') {
        proxy_pass  $solr_server/mba-clubs/select?q=*:*&$commonParams&sort=title_complete%20asc;
    }
    proxy_pass  $solr_server/mba-clubs/select?q=title:$arg_q%20content:$arg_q&$commonParams;
}
```

Figura 29. Petición del índice MBAClubs

Aquí ya podemos observar algo de lógica. Observamos un condicional if, que nos indica que, si el parámetro q que obtenemos es un *, buscaremos con *:*. Este indicador, *:* significa ejecutara una búsqueda predeterminada en todos los campos. En este caso, además de los parámetros comunes, aplicamos una ordenación de tipo ascendente y sobre el campo title_complete. Sin embargo, en el caso de que obtengamos algo diferente en el campo q, buscaremos solamente en el campo title y content, separándolos con un %20 que es la codificación es ASCII para el carácter de espacio. Una vez más añadimos los parámetros comunes, pero aquí no ordenamos.

```
location /search/financial-aid/ {

    set $financialfilter $arg_filter;

    set_by_lua_block $financialfilter {
        if string.match(ngx.var.financialfilter, "nationalities_codes") then
            if string.match(ngx.var.financialfilter, "programs") then
                local a = ngx.re.gsub(ngx.var.financialfilter,"(%20AND%20p)", ")$1")
                 a = ngx.re.gsub(a,"(nationalities_codes)","(nationalities_codes:(All)%20OR%20$1")
                return a
            else
                local a =
ngx.re.gsub(ngx.var.financialfilter,"(nationalities_codes)","(nationalities_codes:(All)%20OR%20$1")
                return a .. ")"
            end
        else
            return ngx.var.financialfilter
        end
    }

    set $financialParams
fq=$financialfilter&omitHeader=true&wt=$arg_format&rows=$arg_result_num&start=$arg_result_start;

   proxy_pass
$solr_server/financial/select/?q=name:$arg_q%20programs:$arg_q%20financial_aid_type:$arg_q%20nationalities:$arg
_q%20description:$arg_q&$financialParams;
}
```

Figura 30. Petición de búsqueda del índice de préstamos

Finalmente encontramos una petición con mucha complejidad. Nginx solo permite el uso de unas reglas muy sencillas, como son el condicional if. Muchas veces esto no es suficiente, aunque al usar OpenResty, podemos aplicar códigos de scripting Lua. En este caso, se requería aplicar una serie de filtros laterales en el front-end, que requerían resultados especiales. El problema principal es que se necesitaba hacer modificaciones en los parámetros de la

petición en función del valor de estos. La solución consiste en utilizar la instrucción set_by_lua_block sobre la variable $financialFilters. Dentro de este bloque de Lua, encontramos diversos condicionales para los diversos casos, pero lo importante aquí es ngx.re.gsub(sujeto, RegEx, Substitución). Gracias a esta función, conseguimos manipular los parámetros de filtros a nuestro gusto utilizando expresiones regulares de una forma muy similar a como funcionan en JavaScript o en Java mismo. Finalmente devolvemos el valor que queda asignado en $financialFilters y este se añade a la petición.

Lua es una herramienta muy poderosa que nos ha ayudado con creces en el desarrollo del proyecto, de forma que podemos afirmar que vale la pena utilizar OpenResty en lugar de Nginx únicamente, simplemente por el hecho de poder utilizar Lua.

9.1.5 TI5 Funcionalidades genéricas de solr

Las funcionalidades que vamos a hablar son funcionalidades que obtenemos a través de componentes que Solr nos proporciona [33]. Hemos utilizado varias de ellas para la implementación de la aplicación y son las que veremos a continuación.

La primera es la elevación. La elevación nos permite mostrar unos resultados concretos para una búsqueda concreta, saltándonos todas las prioridades y pesos. Debemos poner el componente en el Handler de la petición y configurar las elevaciones con un fichero elevate.xml.

```
<elevate>
  <query text="name:* programs:* financial_aid_type:* nationalities:* description:*">
    <doc id="3067" />
    <doc id="3063" />
    <doc id="3065" />
    <doc id="1227" />
    <doc id="3071" />
    <doc id="3069" />
    <doc id="1228" />
    <doc id="1239" />
    <doc id="1229" />
  </query>
```

Figura 31. Ejemplo de query elevada

Tal y como se puede observar, pondremos la búsqueda y los ids de los resultados que queremos que se muestren los primeros cuando se ejecute esa búsqueda. El orden de los documentos tal y como los ponemos en el fichero se conserva en la búsqueda. Este componente es muy útil cuando necesitamos poner unos pesos a nuestra búsqueda, pero los primeros resultados se escogen de forma independiente.

Otro componente, el más útil probablemente, es el de corrección. En este componente le indicamos para que campos vamos a aceptar fallos tipográficos. Podemos indicarle la cantidad de fallos permitidos, por ejemplo, 1 sola letra. Si lo activamos y buscamos “Carera” en vez de “Carrera” los resultados serán iguales que si buscáramos la palabra correctamente. Obviamente, hay que tener en cuenta que hay palabras que si les cambiamos una letra siguen existiendo. Es por eso por lo que los resultados sugeridos por el corrector aparecerán en un clave del JSON aparte.

9.1.6 TI6 Despliegues a producción

La última tarea de la fase de implementación consiste en preparar los entornos y el código para subir a producción. Durante la fase de desarrollo ya se han ido haciendo despliegues al entorno de Preproducción. Realmente, los entornos ya han sido configurados en la TI1. De esta forma, lo único de lo que nos hemos de preocupar es de subir los archivos necesarios a nuestro servidor, pararlo y volverlo a arrancar.

Vamos a dividir el proceso en 3 partes. Por un lado, encontramos la actualización de Nginx, por otro lado, la actualización de Solr y por último la del indexador. Son procesos distintos pero que tienen partes comunes.

En nuestra máquina local, vamos a situarnos en la rama de código GIT donde tengamos los cambios que queremos subir a producción (cabe destacar que este proceso es exactamente el mismo para preproducción).

Una vez estemos en la rama que queramos utilizaremos un comando de Maven para generar un archivo .JAR con el código de nuestro indexador. Este comando es en concreto assembly:assembly [34] y requiere importar un plugin para Maven. Este comando nos permite crear el ejecutable con las dependencias del proyecto dentro, lo que resulta muy útil.

El siguiente paso es acceder al servidor mediante algún programa que nos permita usar el protocolo SFTP [35]. Con este protocolo podemos subir al servidor los archivos que queramos. Por lo tanto, subiremos nuestro ejecutable o en caso de que haya uno anterior por no ser el primero de los despliegues lo que haremos será simplemente substituir el ejecutable. También debemos subir al servidor el archivo de configuración de Crontab como los scripts generados para la indexación, sino no el indexador no se ejecutará nunca y no tendremos datos en Solr. En el caso de querer actualizar la configuración de Nginx, simplemente substituiremos la configuración anterior con la actual. Aplicar cambios en Solr es algo más complejo, pero aun así muy simple.

Debemos cargar tanto nuestro SolrConfig.xml nuevo como el Schema.xml. En el caso de que nos haga falta algún fichero de texto para algún componente lo subiremos también. A continuación, lo que hay que hacer es conectarse al servidor a través del protocolo SSH para poder ejecutar comandos. Debemos borrar el índice, y volver a crearlo utilizando la configuración nueva. La sucesión de comandos sería la siguiente:

```
sudo -H -u solr bash -c '/opt/solr/bin/solr delete -c alumni-chapters'

sudo -H -u solr bash -c '/opt/solr/bin/solr create -c alumni-chapters -d
/opt/solr/server/solr/configsets/alumni-chapters/conf'

sudo -H -u solr bash -c 'sh /opt/solr-indexer/script_alumni_chapters.sh'
```

Figura 32. Sucesión de comandos para reindexar contenido

Ahora tenemos 2 opciones, o bien esperar a la indexación de contenidos que ejecutará el Crontab a la hora determinada o bien lanzar la indexación de contenidos de ese índice en específico. Hay que destacar que para actualizar el Crontab solo hay que substituir el fichero de configuración.

9.2 Sistema de logging y backup

El termino logging se refiere a el hecho de añadir logs en el código, durante su implementación. Un log no es más que una línea de código que nos permite sacar a un fichero cierta información. Por ejemplo, si queremos saber cuánto vale la variable A en un momento determinado, pondremos un log en ese momento que registre su valor. Hay muchos sistemas de logging, pero nosotros hemos utilizado SLF4J [36]. Los logs nos sirven principalmente para detectar errores o comportamientos inesperados, aunque también los usamos mucho a modo de información. Registramos cada indexación que ejecuta el Crontab, si ha sido exitosa, si ha fallado o cuales son los datos que hemos indexado o borrado. De esta forma, vamos a poder detectar la mayoría de las veces si los problemas son nuestros o vienen del origen de datos (por ejemplo, que sean erróneos los datos que cogemos). Así, nos cubrimos las espaldas y podremos dar explicaciones y justificaciones al cliente cuando la aplicación no funcione como debe.

Además, todo sistema necesita hacer backups de vez en cuando. Un backup es simplemente una copia de seguridad. Si algo va mal en algún momento, cargaremos un backup hecho en un momento que sabíamos que la aplicación funcionaba correctamente. En Solr generar backups es muy sencillo, simplemente hemos de ejecutar el comando siguiente:

https://<server>/solr/<index>/replication?command=backup&location=/opt/solr_backup/<index>

Para restaurar esta copia:

https://<server>/solr/<index>/replication?command=replication&location=/opt/solr_backup/<index>

Es un proceso muy sencillo por lo que antes de los despliegues a producción siempre se hará un backup.

10 Documentación

La documentación es una parte vital en el desarrollo de software. En las empresas de hoy en día los desarrolladores van y vienen y es muy fácil que un proyecto empezado por uno lo termine otro. Esta nos sirve para dejar constancia de nuestro trabajo, justificar el porqué de según que decisiones o simplemente soluciones que podremos consultar en un futuro.

Nosotros hemos decidido utilizar dos sistemas: Atlassian Confluence y Postman.

Confluence es un sistema de documentación en el que podemos crear proyectos. Una vez creado nuestro proyecto, vamos a poder crear páginas en él. Nos permite estructurar nuestra información, editarla en línea y todos los participantes del proyecto podrán consultarla. Es una herramienta muy útil y en ella es donde almacenaremos los requisitos de la especificación, diagramas, etc. De esta forma siempre que nos sea necesario podremos consultarlo. Además, crearemos un manual de soluciones que contendrá como hacer backups o información como las IPs de los servidores para futuros desarrolladores que se incorporen al proyecto.

Postman es una herramienta que se utiliza para crear peticiones http hacia nuestro servidor. Sin embargo, algo que no todo el mundo conoce es el hecho de que Postman te permite crear colecciones. Con estas colecciones, agrupamos una serie de peticiones y las podemos compartir con el resto del equipo para que ellos también puedan añadir las suyas o modificar las nuestras. Además, nos permite crear un documento muy visual con todas estas peticiones que resulta muy útil cuando hay que desarrollar como fuente de consulta.

11 Identificación de leyes y regulaciones

Uno de los mayores problemas de hoy en día es la protección de datos. Hay que encriptar la documentación delicada, así como contraseñas, por ejemplo. A nosotros no es un problema que no afecte. Toda la información de la plataforma es pública por lo que no ha de ser encriptada. Además, no guardamos información de ningún tipo de usuario ni persona por lo que es un tema del que no nos tenemos del que preocupar.

El único asunto legal que podría afectar a nuestra plataforma serían las licencias de software. Vamos a ir viendo que licencias requiere cada tecnología utilizada para asegurarnos que no nos genera ningún problema. Cabe destacar que la mayoría de las tecnologías son Open Source. Hay algunas licencias Open Source, como por ejemplo GPLv2, que obligan a que el software desarrollado solo pueda producirse con tecnologías de licencias Open Source. Sin embargo, muchas otras sí que permiten compatibilidad con licencias privadas y que la distribución del software producido sea de forma privada, [37]

El software Open Source está ligado a sus propias licencias. Este siempre se puede utilizar siempre que su distribución y uso sea bajo dichas licencias, por lo que no hay que preocuparse.

IntelliJ	Licencia propia
OpenResty	BSD [38]
Solr	Apache License 2.0 [39]
GIT	GNU-2.0 [40]
Java 8 JDK	BCL [41]

Figura 33. Tabla de licencias

Visto que estamos haciendo un uso correcto de todas las licencias utilizadas, no hay que preocuparse.

Dado que nosotros no mantenemos un control sobre el Front-End de la aplicación no nos hemos de preocupar por el uso de cookies. El uso de cookies suele ser uno de los aspectos legales que hay que tratar ya que el usuario ha de aceptar el uso de estas para una gestión personalizada, de, por ejemplo, la publicidad.

12 Trabajo a futuro

El trabajo a futuro a realizar es básicamente el mantenimiento del proyecto. Una vez el proyecto está en producción, la empresa decide si contratar el mantenimiento con la misma empresa (la mayoría de las veces) o si cambiar. De esta forma, se crea un presupuesto de soporte con un número de horas anuales. El cliente puede gastar estas horas para hacer cambios sobre su aplicación, crear mejoras o arreglar cosas que creen que su funcionamiento no es el que se necesita en ese preciso momento. Una aplicación puede cambiar mucho dependiendo del momento de uso. Sin embargo, cambios muy grandes se facturan como un proyecto aparte.

Para tratar el soporte, se crea un proyecto en la herramienta Jira que nos permite gestionar incidencias, donde el cliente creara dichas indecencias o mejoras y nosotros le daremos una estimación de tiempo. Una vez finalizadas, se desplegarán a preproducción y si el cliente las valida, se subirán a producción.

Una mejora para el proyecto podría ser la implementación de integración y entrega continua para agilizar los procesos de despliegue y el desarrollo de test unitarios y de integración que cubran la mayoría del código, fuera de eso, todo el resto del trabajo a futuro viene dado por el cliente.

13 Planificación y costes finales

En este apartado, analizaremos cuanto hemos tardado realmente en llevar a cabo el proyecto, el porqué de las desviaciones y los costes finales debido a estas.

Tarea	Tiempo estimado	Encargados	Tiempo real
Reuniones	24 horas	Project Manager (100%) Arquitecto (100%)	24 horas
Documentación	120 horas	Project Manager (20%) Desarrollador (70%) Arquitecto (10%)	140 horas
Definición de problemáticas	40 horas	Project Manager (100%)	40 horas
Definición de funcionalidades	40 horas	Arquitecto (100%)	40 horas
Diseño de arquitectura	40 horas	Arquitecto (100%)	60 horas
Prueba de concepto	20 horas	Project Manager (90%) Arquitecto (10%)	20 horas
Preparación de entornos	32 horas	Arquitecto (100%)	32 horas
Creación de índices	80 horas	Arquitecto (10%) Desarrollador (90%)	100 horas
Implementación de Indexador	120 horas	Arquitecto (10%) Desarrollador (90%)	140 horas

Implementación de búsquedas	120 horas	Arquitecto (10%) Desarrollador (90%)	160 horas
Funcionalidades genéricas	50 horas	Arquitecto (10%) Desarrollador (90%)	40 horas
Despliegues	24 horas	Desarrollador (100%)	24 horas
Fase de Pruebas	80 horas	Desarrollador (35%) Tester (65%)	80 horas
Cierre de proyecto	16 horas	Project Manager (80%) Arquitecto (20%)	16 horas

Figura 34. Tabla de costes humanos

Como podemos observar, hay algunas tareas que han variado respecto a la estimación inicial.

Vemos que el tiempo dedicado a reuniones fue estimado correctamente y se han ejecutado las que se establecieron al principio. Hay que tener en cuenta que hubo un contacto constante vía e-mail y Microsoft Teams.

La documentación sin embargo si ha crecido algo. Esta es una tarea complicada de estimar.

Definición de problemáticas, funcionalidades y la preparación de los entornos siguieron la estimación inicial sin embargo el diseño de la arquitectura requirió algo más de tiempo para asegurarnos que era la correcta.

Tanto en los índices como la implementación del indexador se obtuvo un crecimiento de horas, a pesar de que algunos índices tenían configuraciones muy similares, los complicados retrasaron algo las tareas.

La implementación de búsquedas fue la tarea que más se retrasó respecto a la estimación. Configuraciones de OpenResty como en el índice de préstamos

tal y como hemos visto en el apartado de implementación han retrasado esta tarea. El aprendizaje de Lua para poder hacer todas esas modificaciones también influyó en dicho retraso.

Las funcionalidades genéricas se consiguieron acelerar mientras que los despliegues, la fase de pruebas y el cierre siguieron exactamente igual.

Recordemos lo precios de cada rol:

- Project manager: 25.40 euros/hora
- Desarrollador: 19.46 euros/hora.
- Tester: 20.87 euros/hora.
- Arquitecto: 22.89 euros/hora.

Teniendo en cuanta las desviaciones, si hacemos el cálculo de costes de nuevo, exactamente igual que hicimos en las estimaciones en el apartado de estimaciones obtenemos el siguiente coste:

- Recursos humanos recalculados: 26829.91 euros
- Los recursos materiales no han variado: 578.72 euros.
- Los costes indirectos tampoco: 988.77 euros.

Costes finales => 26829.91 + 578.72 + 988.77 = 28397.4 euros.

14 Informe de sostenibilidad

14.1 Resumen

Tras completar la autoevaluación sobre la sostenibilidad de los proyectos podemos resumir esta para analizarla.

Se ve como muchas preguntas se centran en la capacidad de análisis de soluciones existentes, decisión de la solución definitiva y finalmente la optimización de la solución propuesta.

Así pues, creo que en este proyecto no aplica el análisis de soluciones anteriores dado que no había una solución previa. Lo único que podemos analizar son las posibles soluciones del mercado y escoger aquellas más sostenibles.

En cuanto a la decisión de la solución y optimización de esta, definitivamente es una solución sostenible por el hecho de utilizar software libre, que no genera un coste extra y minimiza este, así como la mayoría de las herramientas. Reutilizar los servidores del cliente también lo convierte en una solución sostenible por el hecho de no generar más basura tecnológica innecesaria.

Por último, una debilidad podría ser la falta de estudio con indicadores igual que la poca colaboración social dado que es un proyecto cuyo cliente es una entidad privada y no social.

14.2 Dimensión económica

En cuanto al coste del proyecto, podemos considerar que no es nada desmesurado si tenemos en cuenta el alcance de este. No es un simple buscador si no que centraliza toda la información lo que lo convierte en un proyecto fácilmente escalable a otras fuentes de origen de datos, así como base de datos para futuros proyectos. Esto implicaría una reducción en los costes de futuros proyectos.

Hay que tener en cuenta que es una solución nueva, lo que implica que no se puede optimizar una solución existente.

14.3 Dimensión ambiental

En cuanto a la dimensión ambiental creo que un punto a favor es el uso de los servidores del propio cliente que reduce tanto el coste de tener un servidor nuevo, así como el consumo de este. El impacto de nuestro proyecto será básicamente el consumo eléctrico y de internet de este, totalmente necesario para su desarrollo.

14.4 Dimensión social

Este proyecto supone una mejora social para la universidad que lo utilizará. Esto se debe a que el contenido será más accesible para todos los usuarios de su página web, e incluso teniendo un apartado de profesores y eventos que desde luego tiene un impacto social.

Es un proyecto necesario para nuestro cliente, más por la centralización de datos que por el buscador, pero que al ser tan escalable puede solucionar muchos futuros problemas de forma óptima.

15 Conclusiones

Para concluir, vamos a hacer una introspectiva sobre el transcurso del proyecto para determinar si ha sido un éxito y si hemos cumplido las competencias técnicas asociadas a este.

El estudio previo de los requisitos nos ha permitido definir unos objetivos muy claros y específicos, lo cual es importante para poder solventar el problema. Por eso, el problema ha quedado bien definido desde un primer momento, sabíamos donde estábamos y hacia donde queríamos ir.

En cuanto al diseño, tenemos una aplicación escalable y práctica. Tenemos la oportunidad de hacerla crecer si queremos añadir un índice nuevo o borrar uno no significa ningún tipo de complicación para los desarrolladores.

Las funcionalidades implementadas han sido utilizadas y resuelven los problemas y alcanzan los objetivos propuestos. Así como las estimaciones de tiempo de estas se han cumplido.

A nivel personal, he aprendido a tratar con clientes, así como nuevas tecnologías que me han hecho dar el máximo de mí. Creo que he crecido tanto personal como tecnológicamente por lo que por mi parte quedo contento y satisfecho por el trabajo hecho.

En cuanto a las competencias técnicas:

- **CES1.1:** El sistema se ha desarrollado de zero y está en producción, actualmente en mantenimiento.
- **CES1.2:** Las tencologias se han escogido en función de los requisitos y las espcificaciones.
- **CES1.3:** Se llevó a cabo un estudio de riesgos y contingencias.
- **CES1.4:** Se han utilizado protocolos como Https para la comunicación y se coge información de diferentes puntos de origen.
- **CES1.5 y CES1.6:** Solr en si es una base de datos no relacional la cual se ha escogido dado que cumplia los requisitos y se ha diseñado los esqumas de los índices de forma adecuada. La información ha sido gestionada y modificada por el sistema.

- **CES1.7:** Se ha creado un documento de pruebas gracias a Postman y una documentación extensa. Había una persona dedicada al testing manual y todas las funcionalidades han sido aprovadas por el cliente.
- **CES2.1:** Se llevo a cabo una fase de especificación con toma de requisitos y estos se gestionaron de forma correcta.

En definitiva, el problema ha sido resuelto, está en producción funcionando y el cliente está satisfecho, de forma que podemos afirmar que el proyecto ha sido todo un éxito.

16 Bibliografía

[1] Significados, «Definición de stakeholders,» [En línea]. Available: https://www.significados.com/stakeholder/. [Último acceso: 22 9 2019].

[2] M. Ganesh, «Solr vs ElasticSearch,» [En línea]. Available: https://blog.expertrec.com/solr-vs-elasticsearch-which-is-better/.

[3] G. Pinegar, «Metodologia Waterfall,» [En línea]. Available: https://learn.g2.com/waterfall-methodology.

[4] Verisign, «Certificados SSL,» [En línea]. Available: https://www.verisign.com/es_LA/website-presence/online/ssl-certificates/index.xhtml. [Último acceso: 27 09 2019].

[5] D. Lázaro, «Autenticación HTTP,» [En línea]. Available: https://diego.com.es/autenticacion-http. [Último acceso: 27 09 2019].

[6] T. fácil, «Sistemas Open Source,» [En línea]. Available: https://tecnologia-facil.com/que-es/que-es-open-source/. [Último acceso: 29 09 2019].

[7] G. d. Cataluña, «Convenio colectivo oficinas y despachos,» [En línea]. Available: https://www.ccoo-servicios.es/archivos/catalunya/convenio-oodd-2017-18-cast-dogc.pdf.

[8] Ceolevel, «Salario Project Manajer,» [En línea]. Available: http://www.ceolevel.com/cuanto-cobra-project-manager-descubre-estas-la-media.

[9] Indeed, «Salario desarrollador,» [En línea]. Available: https://www.indeed.es/salaries/Programador/a-senior-Salaries,-Barcelona-CT.

[10] Indeed, «Salario Tester,» [En línea]. Available: https://www.indeed.es/salaries/QA-engineer-Salaries,-Catalunya.

[11] Indeed, «Salario arquitecto,» [En línea]. Available: https://www.indeed.es/salaries/Arquitecto/a-java-Salaries.

[12] Lenovo, «Precio portatiles,» [En línea]. Available: https://www.lenovo.com/es/es/laptops/thinkpad/t-series/T495/p/22TP2TTT495.

[13] P. componentes, «Pantalla DELL,» [En línea]. Available: pccomponentes.com/dell-p2719h-27-led-ips-fullhd?gclid=CjwKCAjwxOvsBRAjEiwAuY7L8q3QZt2udTVGxNkROFngHxG1ho8CU-n7NVkwvkGVV2SQfRnCvpzbwRoCyBcQAvD_BwE.

[14] JetBrains, «Licencia IntelliJ ultimate,» [En línea]. Available: https://www.jetbrains.com/idea/buy/#commercial?billing=yearly.

[15] d. D. Rovisco, «Cálculo de desviaciones,» [En línea]. Available: https://informacionparalaaccion.com/un-truco-para-analizar-las-desviaciones/.

[16] ABC, «www.abc.es,» [En línea]. Available: https://www.abc.es/tecnologia/consultorio/20150216/abci--201502132105.html.

[17] json-ld.org, «Json-ld,» [En línea]. Available: https://json-ld.org/.

[18] rss, «Rss explicada,» [En línea]. Available: https://www.rss.nom.es/.

[19] A. Pointeau, «openclassrooms.com,» [En línea]. Available: https://openclassrooms.com/en/courses/4990961-planea-tu-proyecto-con-uml/4996511-diagramas-de-casos-de-uso.

[20] Apache, «apache solr,» [En línea]. Available: https://lucene.apache.org/solr/.

[21] Apache, «Lucene,» [En línea]. Available: https://lucene.apache.org/.

[22] Nginx, «nginx,» [En línea]. Available: https://www.nginx.com/.

[23] OpenResty, «openresty,» [En línea]. Available: https://openresty.org/en/.

[24] Lucain, «Desde linux,» [En línea]. Available: https://blog.desdelinux.net/cron-crontab-explicados/.

[25] J. Ordóñez, «idento,» [En línea]. Available: https://www.idento.es/blog/desarrollo-web/que-es-una-api-rest/.

[26] Digicert, «digicert,» [En línea]. Available: https://www.digicert.com/es/what-is-ssl-tls-and-https-es/.

[27] GraphEverywhere, «GraphEverywhere,» [En línea]. Available: https://www.grapheverywhere.com/bases-de-datos-nosql-marcas-tipos-ventajas/.

[28] S. Guide, «Apache Lucene Standard Query Parser,» [En línea]. Available: https://lucene.apache.org/solr/guide/7_5/the-standard-query-parser.html.

[29] S. Guide, «Apache Lucene DisMax Query Parser,» [En línea]. Available: https://lucene.apache.org/solr/guide/7_5/the-dismax-query-parser.html.

[30] S. Guide, «Apache Lucene eDisMax Query Parser,» [En línea]. Available: https://lucene.apache.org/solr/guide/7_5/the-extended-dismax-query-parser.html.

[31] S. Guide, «Apache Lucene Tokenizers,» [En línea]. Available: https://lucene.apache.org/solr/guide/7_5/tokenizers.html.

[32] S. Guide, «Apache Lucene Filters,» [En línea]. Available: https://lucene.apache.org/solr/guide/7_5/about-filters.html.

[33] S. Guide, «Apache Lucene Search Components,» [En línea]. Available: https://lucene.apache.org/solr/guide/7_5/requesthandlers-and-searchcomponents-in-solrconfig.html.

[34] Apache, «Maven Assembly Plugin,» [En línea]. Available: http://maven.apache.org/plugins/maven-assembly-plugin/.

[35] Strato, «Que es SFTP,» [En línea]. Available: https://www.strato.es/faq/hosting/que-es-sftp-y-como-se-utiliza/.

[36] SLF4J, «Logger sistem,» [En línea]. Available: http://www.slf4j.org/.

[37] M. C. V. Alcober, «Heraldo,» [En línea]. Available: https://www.heraldo.es/noticias/sociedad/2017/05/31/aspectos-legales-tener-cuenta-desarrollo-vendo-software-1178653-310.html#.

[38] G. Lehey, «freebsd,» [En línea]. Available: https://www.freebsd.org/doc/es/articles/explaining-bsd/article.html.

[39] apache, «Licencia Solr,» [En línea]. Available: https://www.apache.org/licenses/LICENSE-2.0.

[40] Gnu.org, «Gnu operating system,» [En línea]. Available: https://www.gnu.org/licenses/old-licenses/gpl-2.0.html.

[41] Oracle, «Oracle,» [En línea]. Available: https://www.oracle.com/technetwork/java/javase/overview/oracle-jdk-faqs.html.